VEHICLE ROUTING FOR WASTE COLLECTION IN BAGUIO CITY USING PSO-GA

by

Lance Oliver Licnachan

A special problem submitted to the
Department of Mathematics and Computer Science
College of Science
The University of the Philippines
Baguio, Baguio City

as partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science

June 2018

This is to certify that this Special Problem entitled **“Vehicle Routing for Waste Collection in Baguio City Using PSO-GA”**, prepared and submitted by **Lance Oliver Licnachan** to fulfill part of the requirements for the degree of **Bachelor of Science in Computer Science**, was successfully defended and approved on June 25, 2018.

JOEL M. ADDAWE, PH.D.
Special Problem Adviser

The Department of Mathematics and Computer Science endorses the acceptance of this Special Problem as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science .

JERICO B. BACANI, PH.D.
Chair
Department of Mathematics and
Computer Science

Table of Contents

Acknowledgments

I would also like to acknowledge my family and friends for their support in my pursuit to accomplish this study. I would also like to acknowledge the Solid Waste Management Division of Baguio City for entertaining my requests and questions about the current state of waste management in the city.

Abstract

Vehicle Routing for Waste Collection in Baguio City Using PSO-GA

Lance Oliver Licnachan
University of the Philippines, 2018

Adviser:
Joel M. Addawe, Ph.D.

Waste collection services are the key to proper development in any country. In Baguio City, waste collection has been a main concern for the past years as population increases. The waste collection problem was modeled as a Vehicle Routing Problem and a hybrid PSO-GA algorithm was employed to obtain the set of routes that give the minimum amount of travel distance. It was found that the hybrid PSO-GA algorithm was indeed able to solve vehicle routing problems and that the optimal set of routes give the minimum distance of $1,284.830000$ kilometers using 74 vehicles.

List of Tables

List of Figures

Chapter 1

Introduction

Municipal solid waste management is one of the key services held as a foundation in developing urban cities around the world. It is without a doubt that urbanization and development comes with the production of large volumes of waste. Moreover, in densely populated cities, waste management becomes challenging as it is met with increasingly larger amounts of municipal solid waste generated by the increasing population size. Development and industrialization ensues the improvement of the standards of living and an increase in the total amount of disposable income, hence, individuals become encouraged to consume goods and services, thereby resulting in an increase in the amount of waste generated. Couple this with the fact that major economies run on an unyielding cycle of production and consumption, waste generation is likely to gain speed as cities become more industrialized and populated. According to the study conducted by the World Bank, called "WHAT A WASTE: A Global Review of Solid Waste Management", back in 2012, global urban population annually produced about 1.3 billion metric tons of Municipal Solid Waste (MSW) which is expected to grow to about 2.2 billion metric tons in 2025.[19]

According to the WHAT A WASTE,[19] waste management in underdeveloped countries are worse than that of developing ones. This is because there are few to no proper facilities or vehicles that are needed to efficiently handle waste collection. Poorly managed waste can result to the destruction of the environment, and endangerment of public health. If waste is left uncollected, it may lead to sewer flooding, different kinds of pollution (such as soil, air, water etc.), and road blockades. Methane, a highly flammable gas, is produced when garbage is decomposed, hence, uncollected waste can become a source of residential fires. Moreover, toxic chemicals from household materials may seep out of garbage bags and contaminate both soil and water. This endangers both flora and fauna living in the environment. In addition to the effects to public health, mismanaged waste can impact an area's economy. Businesses that do not comply with the regulations imposed by public health and safety get shut down. Uncollected waste can result to a hazardous work environment that can endanger employers, employees, and customers. Garbage can be a breeding ground for diseases which can lead to epidemics that may cripple not only the work flow but also foreign interest. In contrast, a city that has efficient waste management would be able to develop faster as its focus shifts to other community needs such as transportation, health, and education. Given the effects of waste collection mismanagement, all units of the community should improve their own waste management to cope with the problems that are brought about by large amounts of solid waste.

The waste management services industry is a growing business. Developed nations highly depend upon their waste management service providers to move waste out of urban districts and industrial sectors. These services help reduce public health risks and provide a clean atmosphere for a conducive area of business. Normally, people would rather work in a safe and clean environment than at a filthy and odorous one however, people that do work on waste are being paid a generous amount of money. This is because on-site workers risk their lives working with different kinds of waste which may cause them to be exposed to toxic and highly dangerous substances, hence, there must be a good amount of compensation. Not only that but the construction and maintenance of facilities needed for safely dealing with waste are expensive. This is the reason governments allocate a large portion of their annual budget for waste management. According to WHAT A WASTE[19], solid waste management costs will increase from an annual $205.4 billion in 2012 to about $375.5 billion in 2025 to compensate for the projected increase in the amount of waste generated.

Waste, in this context, is defined as matter that is unwanted or unusable. This refers to matter which are discarded after primary use or is deemed defective, or worthless. There are various types of waste but our focus is on municipal solid waste. This is the type of waste that is produced daily from residential, commercial, institutional and industrial sources. The term 'municipal' comes from the fact that it is the duty of the municipality to collect and manage these kinds of waste. A list of what can be considered as municipal solid waste is as follows:

- **Biodegradable waste** which is any form of organic matter that is found in the trash. This type of solid waste are usually composed of leftover food, by products of cooking, agricultural waste (such as lawn clippings, dead leaves, etc.) and paper-based materials.
- **Recyclable materials** which are objects that can be re-used in an alternative way such as glass, bottles, jars, clothes, fabrics, rubber etc.
- **Residual waste** is the type of solid waste that is neither recyclable nor reusable. This may include items that are beyond repair such as broken instruments, shattered glass and ceramics, and used fireworks.
- **Inert** or **nonreactive waste** such as dirt, rocks, construction debris, etc. These items do not react, chemically or physically, and hence, do not decompose.
- **Electrical and electronic waste** by its name, are discarded electrical and electronic devices or components such as appliances, light bulbs, mobile phones, television sets, etc.
- **Composite waste** which are composed of two or more constituent materials such as toys, tetra packs, clothing, fiber glass, etc.
- **Hazardous waste** which poses health risks such as paints, batteries, aerosol sprays, and fertilizers
- **Toxic waste** which are poisonous such as pesticides, herbicides and fungicides
- **Biomedical waste** which may contain infectious materials such as expired pharmaceuticals, used medical equipment, used tissues, etc.

Waste collection includes gathering, transporting, and disposing of solid waste and recyclable materials. Waste collection involves deploying vehicles that collect and transport the waste from communities to facilities that receive, sort, and process the waste. Processing waste may be in the form of incineration, rapid degradation, segregation, resource recovery, energy recover, etc.

Waste can be collected in several ways. House-to-house collection is done by individually visiting each house and collecting the garbage straight from the source. Communities can opt for gathering their garbage at certain designated locations or community bins in the neighborhood. Another way is through curbside pick-ups where households leave their garbage directly in front of their houses to be picked-up by waste collection vehicles at a particular time. Households can also volunteer to personally deliver their garbage directly to disposal sites. The local government can opt for waste collection services with private companies that have specialized facilities and vehicles that can handle the job.

Before collection, households can be asked to segregate their garbage into specified categories. These categories might include, wet and dry, biodegradable and non-biodegradable, reusable, recyclable, paper, glass, plastics, aluminum etc. The quality of waste segregation can determine efficiency and effectiveness of waste processing. Certain waste can be re-segregated by a machine-sorter or by designated personnel called 'pickers'. The reusable quality of an object can be determined by the machine or by personally inspecting the object. Waste such as bear bottles and plastic-containers can be taken out and deemed reusable. Waste such as broken plastic frames or metals can undergo secondary use through remelting and remolding. Papers and plastics can be recycled and re-purposed for a different niche. Biodegradable waste can be converted to compost and degraded through anaerobic digestion. The rest can be piled into landfills or disintegrated using incinerators. Landfills, however, come with other sets of problem such as health-care, contamination, pollution, land use etc. On the other hand, incinerating waste produces greenhouse gases which may raise health care issues. There even exists materials that are immune to incineration such as soil and rocks. Toxic waste disposal is also an arduous task since harmful chemicals are involved. One cannot just dump them in a field and hope for the best. The more developed the facilities and methods are, the better the waste management.

Each of the different collection options presented above have different ways of computing cost. Costs in waste collection may involve staff salaries, vehicle purchases, fuel costs, construction of buildings and infrastructure, fuel expenditure, maintenance, land use and purchase, business contracts, environmental and health care expenditures, social acceptability, etc. We have seen that waste collection services is an expensive service to develop and maintain. It is therefore crucial to find ways to minimize the costs involved whilst not compromising efficiency. Therefore, we look to finding he most efficient set of waste collection routes in order to minimize waste collection expenses.

The routes of waste collection vehicles are, conventionally, manually determined by drivers on-site. Data collection and surveys can help analyze the conditions that affect vehicle routing such as traffic severity and road inclination. However, the use of such methods produce inflexible routes because they are instance-based data. In the recent years, with the development of commercial interactive navigational tools such as Google Maps, Waze and Global Positioning Systems, vehicle routing has become a hot topic. Researchers have used similar kinds of systems to improve delivery and collection services. In order to solve these kinds of problems, a vehicle routing problem model is patterned after the specific problem. There are various methods that can be used to solve this kind problem.

The **Vehicle routing problem (VRP)** involves the deployment of a fleet of vehicles which are expected to service a given set of customers. Solutions to the vehicle routing problem are a set of routes for the fleet of vehicles wherein the set gives the minimum amount of traveling cost. VRP is a generalization of the **Traveling Salesman Problem (TSP)**. The traveling salesman problem is description is as follows:

1. A salesman needs to visit every city in a set of cities.
2. He/she does not care about the order of visitation as long as he/she gets to visit each one along the way.
3. He/she must begin and end at the same city
4. Each city is connected to other close by cities, by airplanes, or by road or railway. Hence, each of the connections between the cities has one or more costs attached depending on the availability of transportation means.
5. The cost describes how "expensive" it is to travel from one city to another. This may be in the form of the cost of an airplane ticket or train ticket. Perhaps the cost may be given by the length of the distance or span of time needed to travel from one city to another
6. The salesman wants to keep both the travel cost and the length of the distance he travels to a minimum.

The aim is to generate a path or a sequence of cities that lets the salesman pass through all cities exactly once before returning to the starting city, spends the minimum amount of travel expenses and the travels shortest possible distance. The problem is mostly concerned about generating the best sequence of N cities among the $(N-1)!$ permutations. As we can see, the amount of permutations rapidly increases as the number of cities is increased. $N-1$ because we always start at a given city therefore, we can say that the salesman has already visited that city.

The vehicle routing problem is typically the same problem however, there are more than one salesman. The load of the single salesman is distributed in order to save other costs (i.e. time) and maximize man power. VRP is a combinatorial problem that deals with arranging multiple specified locations along a single or multiple path(s) that gives the smallest amount of transportation cost while satisfying known constraints. VRP is usually concerned with the minimization of the temporal and/or geographical aspects of traveling along road networks while accommodating the most amount of customer demands along the way. Customers are usually distributed at different locations in the real world. In a capacitated VRP, each customer has a certain amount of demand that utilizes the maximum capacity of the vehicles servicing them. Vehicles neither collect nor delivery more than their capacities. VRP models may also include minimizing the number of vehicles needed to satisfy the total customer demand.

Municipal waste collection VRP involves vehicles collecting municipal waste at customer locations or community bins, and disposing the load at disposal sites. Vehicles start at a depot where they are parked. They are then deployed and travel along their respective routes while collecting waste. When full, the vehicle then moves to a disposal site to unload the waste collected. The vehicle may then either continue along it's path or return to the depot. Waste Collection Vehicle Routing Problems may impose that waste must either be completely or partially collected by a vehicle. Complete collection invokes the rule in VRP that customers are only serviced once by any vehicle. On the other hand, partial collection implies that customers can be serviced more than once by one or more vehicles.

The next sections of this paper are as follows. Chapter 2 involves the discussion of studies conducted by researchers in the previous years. Chapter 3 is a discussion of the method used in order obtain the set of routes. Chapter 4 is where the results from the tests done with the method in chapters 3 is analyzed. Chapter 5 is where the results are summarized and conclusions are stated.

1.1 Background of the Study

According to the National Solid Waste Management Commission, about $37,000$ tons of municipal solid waste (MSW) are produced in the Philippines. Based from the available data from 2008 to 2013, most of the total municipal solid waste in the Philippines comes from the residential sector at 56.7%, while the contributions of the commercial, institutional, and industrial sectors are 27.1%, 12.1% and 4.1% respectively. The municipal solid waste is mostly composed of biodegradable waste at 52.31% while recyclables, residual and special wastes contribute 27.78%, 17.98% and 1.93% respectively. Biodegradable waste consists of kitchen or food waste as well as yard or garden waste. Recyclable wastes consists of plastic packaging, paper and cardboard, metals, glass, textile, leather and rubber. Special waste consists of household health-care waste, waste electrical and electronic equipment, bulky waste and other hazardous materials. Residual waste is composed of the waste that is neither biodegradable, recyclable, nor special waste. This is the type of waste that is sent to landfills. It was projected that the amount of MSW is to increase to about $40,000$ tons in 2016 from $37,000$ in 2012. Out of the 16 regions, the Cordillera Administrative Region (CAR) contributed about 1.66% in 2012.

In 2015, the City of Baguio in the Cordillera Administrative Region conducted a Waste Analysis and Characterization Survey (WACS)[13] where they investigated the output of garbage in the city. It was found that the residents of the city produced 402 tons of mixed waste daily, 41.67% being biodegradable, 33.78% recyclables, 21.41% residuals for recycling, 2.74% residuals for disposal and 0.41% special wastes. Most of the generated waste came from the commercial sector at 60.44%, the contribution of the residential, institutional and industrial sectors are 35.16%, 3.53% and 0.86% respectively. Baguio City, at its core, is a place where farmers from both the surrounding mountains and valleys take their crops to be sold hence, the reason for the commercial sector generating a majority of its waste.

In relation to the WACS, the city drafted a 10 year solid waste manage ment plan as required by Republic Act 9003 or the Ecological Solid Waste Management Act of 2000. Using the WACS, it was projected that the population of Baguio City would climb to about $398,215$ in 2025 from $337,798$ in 2015 and the daily generated waste would rise to about 522 tons from 402 tons. Inclusive of this 10 year solid waste management plan, several facilities are to be constructed for the recovery of resources from municipal solid waste. The plan was approved on 2017, the city is now en route to establishing and developing several waste collection facilities for the next ten years, namely, a centralized materials recovery facility, an engineered sanitary land-fill, an anaerobic digester, a waste-to-energy plant, Environmental Recycling System machines, a health and medical waste treatment plant, and a special waste treatment plant.[31] As of 2018, Baguio City has 14 functioning waste collection trucks, two of which are used as a quick response team in cases of emergencies. The city has purchased 4 more vehicles this last January. This move of purchasing vehicles was said to boost efficiency and to keep-up with the growing tourist influx during the weekends, holidays and the incoming summer vacation.[32] The 14 waste collection vehicles are responsible for servicing the 129 barangays (villages) inclusive of the Central Business District. Vehicle compartments are partitioned such that there is segregation between residual and biodegradable waste. Recyclable materials are usually dealt with by the barangays (villages) who hire personnel to sort the garbage and take out reusable, recyclable materials. The drivers follow a 5-day schedule, the other two days of the week are given as rest days. In each of the five days, they are to service a set of 2-5 barangays. The drivers are set to work 9 hours each working day. Currently, the drivers are the ones who select their routes; they try to avoid traffic in order to attend to each designated collection site where the residents of each barangay pool their waste. All vehicles start at the Eco-Waste Recovery Services-Material Recovery Facility (ERS-MRF) at Barangay Irisan where the drivers sign in for the day. Garbage is collected until vehicle capacities are reach. Waste is returned to the ERS-MRF for final sorting. The residual waste is then transported to the Garbage Transfer Station at Barangay Dontogan. Biodegradable waste is composted while recyclable and reusable materials are sold. It is important to note that there are no specific time windows when each collection site is visited because there are too much variables that can affect collection time such as traffic conditions, weather conditions, amount to be collected, etc.

Indeed, the city is doing it's part to reduce the carbon footprint and employ better waste management by employing the no plastic policy or the "Plastic and Styrofoam-Free Baguio Ordinance" of 2017. This city ordinance regulates the sale, distribution and use of plastic bags and styrofoam in the city. Instead of plastics and styrofoam containers, vendors are encouraged "to provide or make available to customers for free or for a cost, paper bags or reusable bags or containers made of paper or materials which are biodegradable, for the purpose of carrying out goods or other items from the point of sale.[28]

In line with these city policies and activities, we study the current efficiency of the routing and scheduling of waste collection vehicles. Our motivation is to help the community.

1.2 Statement of the Problem

We want to find a way to reduce traveling expenses of waste collection vehicles in Baguio City. It was observed that the cost of waste collection is large for any developing country. Baguio City, for over 10 years, has spent over PHP 1 billion for hauling the city's solid waste to the sanitary landfill in Capas, Tarlac. This involves the operation of collection, transportation, and disposal. In 2017, City Budget Officer Leticia Clemente estimated the annual garbage disposal expense at PHP 100 million, where PHP 80 million of which is spent for hauling and tipping fees in Capas, Tarlac; and for personnel, garbage trucks, and other operating expenses.[21]. The city council wants to reduce annual solid waste disposal expenses so that it can be allocated elsewhere. We approach the problem by modeling the waste collection problem into a vehicle routing problem wherein the goal is to obtain the minimum travel distance required to collect waste and transport it back to the city's Eco-Waste Recovery Services-Material Recovery Facility (ERS-MRF) at Barangay Irisan for sorting before it is transported to Capas, Tarlac. A hybrid Particle Swarm Optimization - Genetic Algorithm (PSO-GA) proposed by Harish Garg[14] will be used to solve the vehicle routing problem.

1.3 Objective of the Study

1.3.1 General Objective of the Study

The general objective of this study is to identify which processes and sections of Baguio City's waste management can be optimized in order to reduce the cost involved in waste management.

1.3.2 Specific Objective of the Study

We identify that both sorting and waste collection are processes that can affect the cost of waste management. We know that the city has implemented its own policies on waste segregation therefore, we look to waste collection. Specifically, we find a way to reduce the operational cost of waste collection vehicles. Moreover, we aim to obtain a set of vehicle routes that give the minimum amount of distance while also determining the minimum number of waste collection vehicles needed for the job.

1.4 Significance of the Study

The results from this study will allow the determination of the possible routes that may minimize the cost of travel among waste collection vehicles in the City of Baguio. Hence, we directly reduce the cost of travel expenses in waste collection. Determining the number of vehicles required to accomplish the job may give incite as to the scale and severity of the waste collection problem in the city. The robustness of the algorithm used in this study will also be determined by the results obtained. Since the hybrid PSO-GA[14] is designed to solve constraint optimization problems, we expect that the algorithm will produce good results for the vehicle routing problem which is another kind of constraint optimization problem compared to that of the engineering design problems Harish Garg used to test the algorithm.

1.5 Scope and Limitation

The study is limited to generating a set of routes involved in waste collection in Baguio City. In order to model the waste collection problem, each of the 129 barangays (villages) were taken as collection sites since there was no available data on the specific locations of the community bins in each of the barangays. This is because the specific collection site locations vary depending on accessibility, road orientation, and public health acceptability. The ERS-MRF at Barangay Irisan was identified as the location where each waste collection vehicle starts and returns when the vehicle is full. Given that there exists no exact time table as to when vehicles are required to visit each barangay, we opt only for a Capacitated Vehicle Routing Problem to model the waste collection problem. The real distances between the ERS-MRF and each barangay as well as the distances between each barangay were obtained through Google Maps©. A map of all 129 barangays are shown on figure 1.1. The red house marker indicates the Irisan ERS-MRF, the yellow flag indicates the Dontogan Transfer Station and the blue markers are the 129 barangay markers. A list of the barangays and their respective markers are seen on table A.1

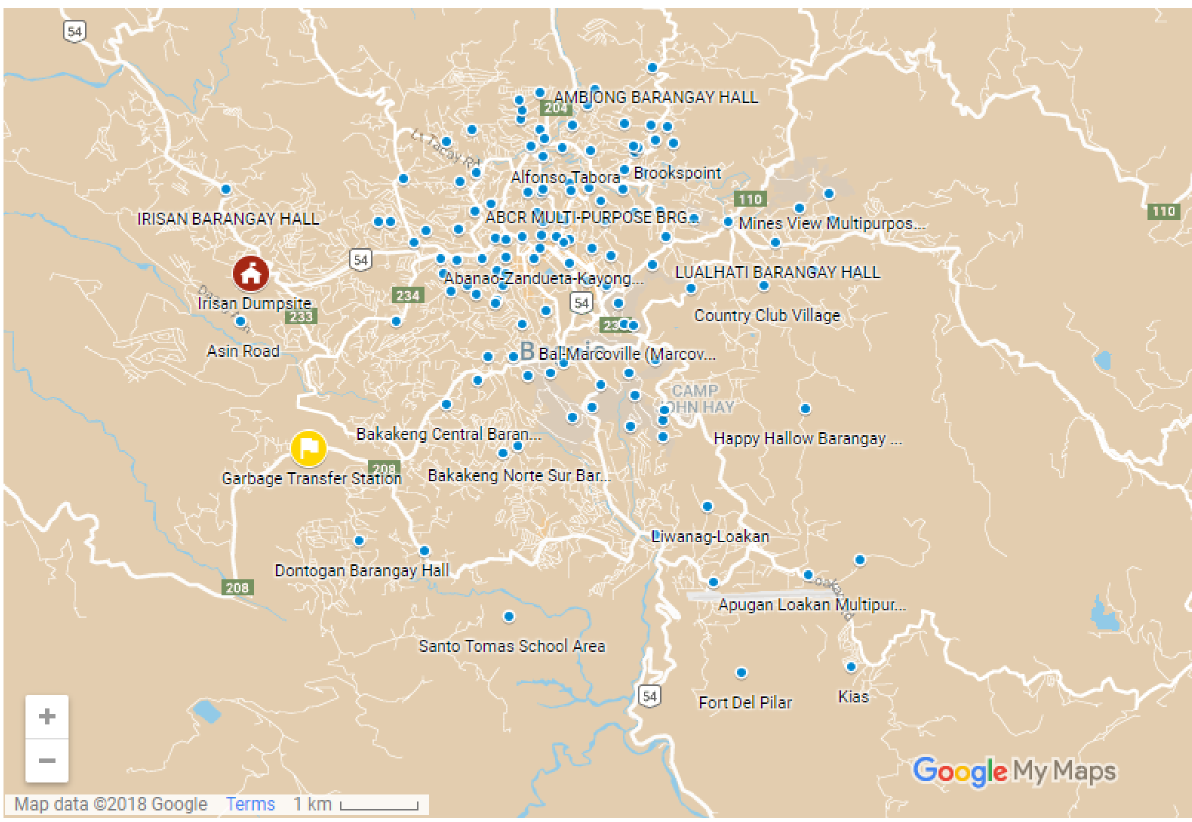

Figure 1.1: A Map Showing the Locations of the 129 Barangays and the Irisan ERS-MRF

Chapter 2

Preliminaries

We first define some terms and notations we will be using throughout this document.

2.0.1 Definitions

1. An **algorithm** is a sequence of unambiguous instructions for solving problems. Every step is clear and concise, no instruction should be interpreted more than one way.

2. **Optimization** is a mathematical technique used for solving the maximum or minimum value of a function or system of equation. In a broader sense, it is a technique used to solve for the optimum solution to a particular problem. Optimality refers to obtaining the best possible form or functionality in the sense that it is more than sufficiently efficient given a set of resources. This involves meeting an expected result with high accuracy and precision such that specifications and limitations are also achieved.

3. An **optimization algorithm** is a process followed in finding the best or most efficient solution to a given problem.

   In order to solve a problem, we must create a model that embodies the essence of the problem. In this sense, the model must be created in such a way that we can approach it through computable means.

4. An **Objective Function** or **Fitness Function** is the mathematical equation that is modeled after the problem such that, satisfying the function will satisfy the given problem. The objective function is important because it will determine the computability and complexity of the problem as well as the approach taken, in this

case, the algorithm and its implementation. Optimization problems aim to obtain the minimum and/or maximum of certain properties related to some object. The output of the objective function dictates whether or not a specific input is not only a solution but also the most optimal one. We say, it is a 'fitness function' because it measures the capability and efficiency of the input in solving the problem.

5. **Design Variables** are the input to objective functions. We say 'design variables' because these sequence of numbers are being used to test and determine the quality of the output. The algorithm is tasked to manipulate the values of these variables in order to get the optimal solution. In tackling real world problems, design often involves a huge amount of data collection through trial-and-error. Our variables are associated to the factors which undergo changes in values during the trial-and-error processes. The data collected should give the efficiency or numerical score of the given design variables.

6. A **heuristic** is a technique designed for solving a problem when classical methods are too slow for finding approximate solutions or when classical methods fail to find any exact solution at all. These classical methods are those that use mathematical identities, properties, and theorems to prove, show, derive or systematically find solutions to problems. The objective of a heuristic is to produce a solution within a reasonable amount of time such that the solution is acceptable enough to the implementor. Although time may not be the only factor that may be taken in consideration, it is the most commonly used factor in differentiating the quality of heuristic approaches.

7. **Metaheuristic** is a high-level procedure to find, generate or select a heuristic that may provide a sufficiently good solution to an optimization problem. Since we are dealing with optimization, finding the fastest and most efficient way to solve the problem is considered to help in finding solutions.

8. An **evolutionary algorithm** is a generic population-based metaheuristic optimization algorithm. It uses mechanisms inspired by biological evolution such as reproduction, mutation, recombination and selection. Candidate solutions to the

optimization problem play the role of individuals in a population, and the fitness function determines the quality of these solutions. We say 'candidate solutions' because all of these individuals may give an acceptable solution but not all of them give the best solution. Evolution of the population takes place after repeated applications of the mentioned operators. We say 'evolution' because members of the population changes or are somewhat different as time progresses or as the population shifts from one generation to another.

9. A **simulation** is a computational model that imitates real world situations and processes. These usually involved an implementation of mathematical equations that employ stochastic variables for a more 'lifelike' appearance.

10. A **stochastic variable** is a variable whose value is a random number usually taken from a uniform distribution from 0 to 1. That is, every number between 0 and 1 has an equal chance to be selected.

11. **Natural Selection** is the process by which organisms with better attributes adapted to the environment tend to increasingly survive and transmit their genetic characteristics through generations. **'Survival of the fittest'** is a phrase by Charles Darwin that describes the mechanism of natural selection. It is best understood as survival through reproductive multiplicity. That is, the more survivability an individual has, the more it is likely to reproduce, hence it's genes are more likely to be transmitted to the next generation.
    In natural selection, there is a variation on traits that is to say that individuals have differences in certain attributes such as height, length, shape etc. It is important to note that not all individuals reproduce to the full potential because the environment has a certain limit to the number of creatures it can sustain. The passing of characteristics or traits from one generation to another is called **heredity**. The more advantageous traits is more commonly passed on and retained because they help the individual or group to survive.

12. Gregor Mendel is known as the father of modern genetics. He discovered the mechanics of heredity or how traits are being passed down from parents to offspring.

During cell division, thread-like structures located inside the nucleus of animal and plant cells called **chromosomes** are replicated. These chromosomes contain the genes which dictate the attributes of the individual. They tell how the body is to be built and how it functions.

13. **Recombination** or **Crossover** is the rearrangement of the genetic material by exchanging the same gene subsegments of two chromosomes (one from each parent) which allows for the creation of a new individual that has characteristics similar to those of the father and of the mother. Note that the exchanging process may occur in multiple areas of the strands.

14. **Mutation** on the other hand is the alteration of genes resulting from an error during replication. This results in unique characteristics that may be new from the gene pool of the previous generation. Mutation may be good or bad for the individual but this phenomenon has a low chance of occurring naturally for every generation.

15. **Robustness** is the balance between efficiency and efficacy necessary for the survival in many different environments. For Algorithms, this translates to consistent efficiency under different problems areas such that there is little to no change in the process. This means that there is less cost for redesigns. Note that nature is the best example in terms of robustness. It tries to maintain and cope with the many different changes that occur everyday. Hence, we have evolutionary algorithms as stated above.

16. **Exploration** is the capability of the algorithm to search solutions in parts of the subspace it has not yet taken into consideration. **Exploitation** is the capability of the algorithm to utilize known data in searching for solutions in the search space.

17. A **set** is a collection of well defined objects. In this document, we will talk about sets as a collection of numbers that represent objects. A set is usually denoted by braces ('{' and '}') and capital letters (A,B,C,D,...) (ex. $A = \{1, 2, 3\}$). In a set, the order of enumeration and repetition of numbers do not matter. That is, $A = \{2, 3, 3, 2, 1, 1\}$ is equal to $A = \{1, 2, 3\}$.

18. An object is considered an **element** (denoted by $\in$) of a set if it belongs to the set. Using our previous example, we say that 1 is an element of A ($1 \in A$) but 4 is not an element of A ($4 \notin A$). There are two ways of declaring membership of sets,

    (a) (a) **roster method** where we define all the elements included in a set by listing or enumerating all of them; and

    (b) (b) **rule method (set-builder notation)** where we define all the elements included in a set using their properties.

    An example of the rule method is $A = \{x \text{ is a natural number}, x < 4\}$ which can also be written as $A = \{x|x \in \mathbb{N}, x < 4\}$, to be pronounced as "the set of all x, such that x is an element of the natural numbers and x is less than 4". The vertical bar ('|') is usually pronounced as "such that", and it comes between the name of the variable you're using to stand for the elements and the rule that tells you what those elements are.

19. **Cardinality** of a set is the number of unique appearances of elements in a set. Cardinality is denoted by two vertical bars ('|') separated by the set name such as '$|A|$'. That is, using our example, the cardinality of A written as $|A|$ is 3 because A has unique elements $1, 2$ and 3.

20. A set without elements is called the **null** or **empty set** (denoted by $\varnothing$) that is, $\varnothing = \{\}$. Therefore $|\varnothing| = 0$.

21. A set with infinite elements is called an **infinite set**, $F = \ldots, 1, 2, 3 \ldots.$ and $|F| = \infty$.

22. A **countable** set is a set with the same cardinality as some subset of the set of natural numbers $\mathbb{N}$. A countable set is either a **finite** set or a **countably infinite** set, nevertheless, the elements of a countable set can always be counted one at a time and, although the counting may never finish, every element of the set is associated with a unique natural number.

23. A **Venn Diagram** is a visual representation of the relationships of sets.

24. We say that A is a **subset** of B (written as $A \subseteq B$). If all elements of A are also elements of B. If $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4, 5\}$ then $A \subseteq B$. However if we have the set $C = \{1, 2, 3, 6\}$, $C \nsubseteq B$ because $6 \notin B$ but $A \subseteq C$. A venn diagram of the relationships of A, B and C are shown on figure 2.1.

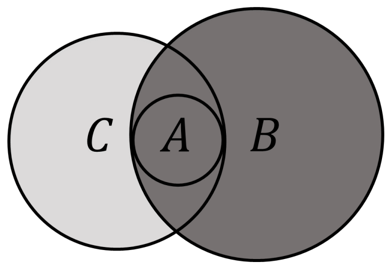

Figure 2.1: A Venn Diagram Showing the Relationship of A, B and C

25. If we have $A = \{1, 2, 3\}$ and $D = \{3, 4, 5, 6\}$, then the **Union** of A and D (written as $A \cup D$) is the set containing all elements of A and D. That is, $E = A \cup D = \{1, 2, 3, 4, 5, 6\}$. A venn diagram showing $A \cup D$ shaded in gray is shown on figure 2.1.

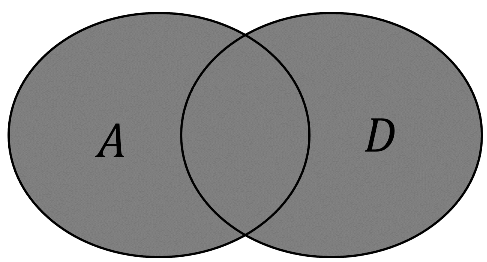

Figure 2.2: A Venn Diagram Showing $A \cup D$

26. If we have the same sets A and D, then the **Intersection** of A and D (written as $A \cap D$) is the set containing all the common elements of A and D. That is, $A \cap D = \{3\}$. A venn diagram of showing $A \cap D$ shaded gray is shown on figure 2.3.

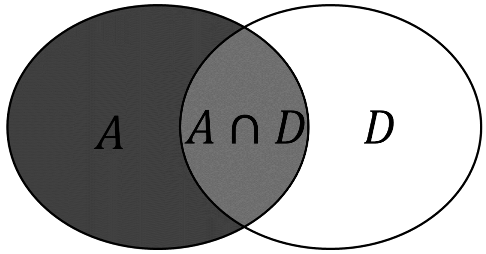

Figure 2.3: A Venn Diagram Showing $A \cap D$

27. If we have the same set A, then the **Complement** of A written as A' or $\bar{A}$ is the set containing all the elements that are not in A. That is $\bar{A} = \{x | x \in \mathbb{N}, x > 3\}$. A venn diagram of showing $\bar{A}$ shaded gray is shown on figure 2.4.

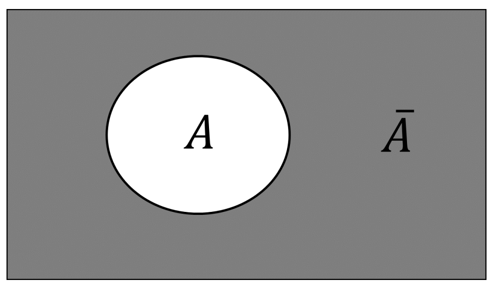

Figure 2.4: A Venn Diagram Showing $\bar{A}$

28. If we have the same sets A and D, then set **Difference (subtraction)** is defined as $A - D$ or $A\ D$ which consists of elements in A but not in D. That is, $A - D = 1, 2$. A venn diagram of showing $A - D$ shaded gray is shown on figure 2.5.

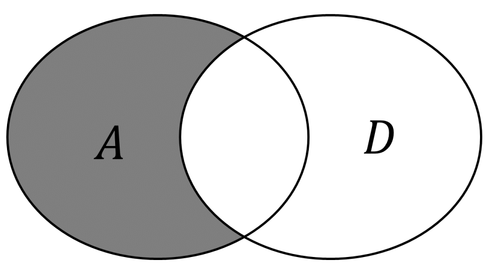

Figure 2.5: A Venn Diagram Showing $A - D$

29. In mathematics, numbers are grouped in sets and subsets.

(a) We first have the smallest subset, the set of **Natural** or **Whole Numbers** ($\mathbb{N}$) which is the set of counting numbers, $\{0, 1, 2, 3, 4, 5\ldots\}$.

(b) The next subset is the set of **Integers** ($\mathbb{Z}$) which is the set of natural numbers and their negatives $\{\ldots$-4, -3, -2, -1, 0, 1, 2, 3, 4, $\ldots\}$.

(c) Next are the **Rational numbers** ($\mathbb{Q}$) are the ratios of integers, also called fractions, such as $\frac{1}{2}$, $\frac{-10}{56}$ etc.

(d) Next are the **Irrational Numbers**, numbers that are not included in the rational number set such as radicals or roots (ex. $\sqrt{5}$) and numbers having infinite non-repeating decimal places such as π.

(e) Finally, the set of **Real Numbers** ($\mathbb{R}$) which consists of both rational and irrational numbers.

(f) Other than the real numbers, we have the **Imaginary numbers** ($\mathbb{I}$) which are the numbers that have negative squares. These numbers are involved with the number $i = \sqrt{-1}$.

(g) The set containing all numbers is called the **Complex Number** ($\mathbb{C}$). This set is the union of both real and imaginary numbers. These numbers are usually represented by the sum of a real and an imaginary number (ex. $1 + i$).

A Venn Diagram of the number sets is given by figure 2.6 .

30. A **solution space** the set of all possible values of an optimization problem that satisfy the problem's constraints, potentially including inequalities, equalities, and integer constraints. This is the initial set of candidate solutions to the problem, before the set of candidates has been narrowed down.

31. **candidate solutions** are potential solutions to problems.

32. A **sequence** is a collection of objects wherein the oder of enumeration is important (ex. a list). Unlike a set, the same elements can appear multiple times at different positions in a sequence, and order of which the elements are enumerated matters, that is if we have two sequences $(1, 2, 3)$ and $(3, 2, 1, 1)$, $(1, 2, 3) \neq (3, 2, 1, 1)$. A

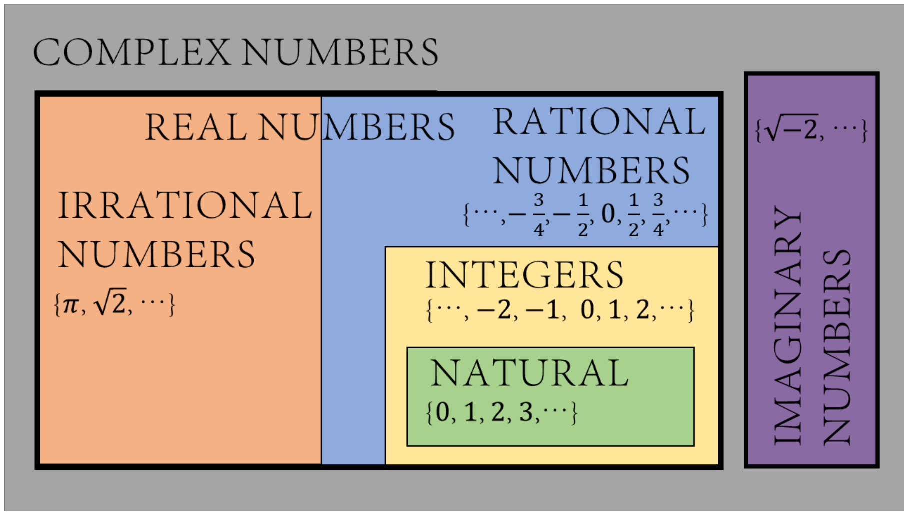

Figure 2.6: A Venn Diagram Showing the Relationship of Number Sets

sequence is usually denoted by parentheses ('(' and ')'), for example, the famous Fibonacci sequence is given as $(0, 1, 1, 2, 3, 5, 8, \dots)$. Mathematical objects, functions or relations are usually described as sequences.

33. The elements of a sequence are called **terms**.

34. The number of elements of a sequence is called the **length** of that sequence.

35. A sequence may be **finite** in length (ex. $(1, 2, 3, 4, 5)$) or **infinite** (ex. $(1, 2, 3, \dots)$) as in sets.

36. Similar to sets, we can define inclusion to a sequence by:

    (a) The **roster** method, generating all its elements, we must be sure that the sequence is finite.

    (b) In case that the sequence may be infinite or has too many elements to list, then we use a **rule**. An example is 'the sequence of alternating 0's and 1's, starting with a 0', $(0, 1, 0, 1, 0, 1, \dots)$.

    (c) We can also use a **formula**. For example, the sequence generated by $(a_n)_{n\in\mathbb{N}} = 2n + 1$ is the sequence of odd numbers starting from 3, $(3, 5, 7, 9, \dots)$.

37. In order to specify which element is being called, we say "**the** n^{th} **term**" of a sequence. For example, given the same sequence $(a_n)_{n\in\mathbb{N}} = 2n+1$ if we want to know the 3rd element of the sequence, we write '$a_3 = 7$', we say "the third term of the sequence is the number 7".

38. A **permutation** is related to the act of arranging items of a set into some sequence or order. The number of all possible arrangements of a set of N items is given by $N!$. If we have the set $A = 1, 2, 3$, the permutations of set A is given as follows:
    - $(1, 2, 3)$
    - $(1, 3, 2)$
    - $(3, 1, 2)$
    - $(3, 2, 1)$
    - $(2, 3, 1)$
    - $(2, 1, 3)$

39. A **point** is a location. It has neither width nor length, even though it is visually represented as a dot for reference.

40. Locations are usually made up of a sequence of numbers called **coordinates**.

41. A **line** is one-dimensional, having length but no thickness. A line is composed of infinite points as it extends infinitely in both directions however, two points are enough to define a line. For example, if we are given two connected points A and B, then make-up the line $\overleftrightarrow{AB}$.

42. A **real number line** is a line wherein each point is associated to some real number $r \in \mathbb{R}$ This makes sense because the set of real numbers is infinite. Since each point is represented as a real number, the coordinate of any point on the line is given by a real number. A visual representation of a real number line is shown on figure 2.7. As previously stated, if we want to know where a point is on the line, we simply tell what number the point represents. Hence, we also know the distance from which the point is located from our reference point, 0.

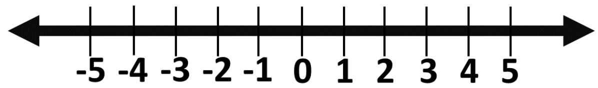

Figure 2.7: A Real Number Line

43. A part of a line that has defined endpoints is called a **line segment**. A line segment as the segment between A and B is written as: $\bar{AB}$. Two lines that meet at a point are called **intersecting lines**. Perpendicular lines are two line that form a 90 degree angle.

44. We say that a set of points are **collinear** if there is a line that passes through all the points.

45. A **plane** is a two-dimensional surface. Ruled and spanned by two independent perpendicular lines. A plane is defined by three non-collinear points.

46. A **coordinate plane** is a plane that is spanned by the real number lines, x-axis and y-axis hence, it is also known as the space R^2. Each point on this plane represents a pair of coordinates (x, y). We usually assign the first number, x, for the distance on the x-axis and the second number, y, for the distance on the y-axis. A coordinate plane is shown on figure 2.8. As we can see, the black point is said to be located at (1,4) this means that it is 1 unit away from (0,0) on the x-axis and 4 units away from (0,0) on the y-axis.

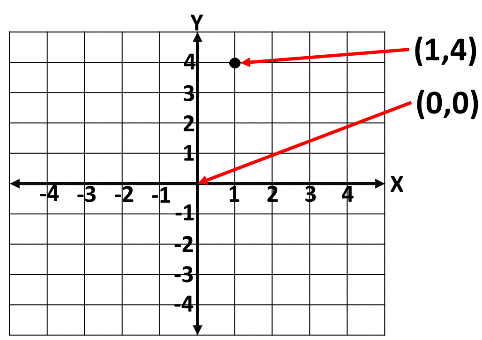

Figure 2.8: A Coordinate Plane

47. A **Vector** is a quantity having both magnitude and direction. In a coordinate plane, it is represented by an arrow as shown in figure 2.9. We can see that vector $a =< 1, 1 >$ is 1 unit to the right of the point (0,0) and 1 unit above the point (0,0). A vector is mainly composed of two points in N dimensions, represented by the points on its tail and its head but it these two points are arbitrary because vectors are only concerned with magnitude and distance but not location. Magnitude is visually represented in length. Direction, one the other hand, is visually represented by the arrowhead. A vector is usually given in the form $< x_1, x_2, x_3, \dots x_n >$ where each component x_i is the absolute numerical distance between two points in dimension $i \in (1, 2, \dots, n)$. When representing vectors in two dimensions, it is broken down into two parts, x and y components. The x component is the horizontal length while the y component is the vertical length. The vector's magnitude ($|a|$) is given by the 2D Pythagorean theorem: $|a| = \sqrt{x^2 + y^2}$ where x and y are its x and y components. In higher dimensions, the same representations follow and the Pythagorean theorem for higher dimensions are used. $|a| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ where each x_i is the component of the vector in dimension $i \in (1, 2, \dots, n)$.

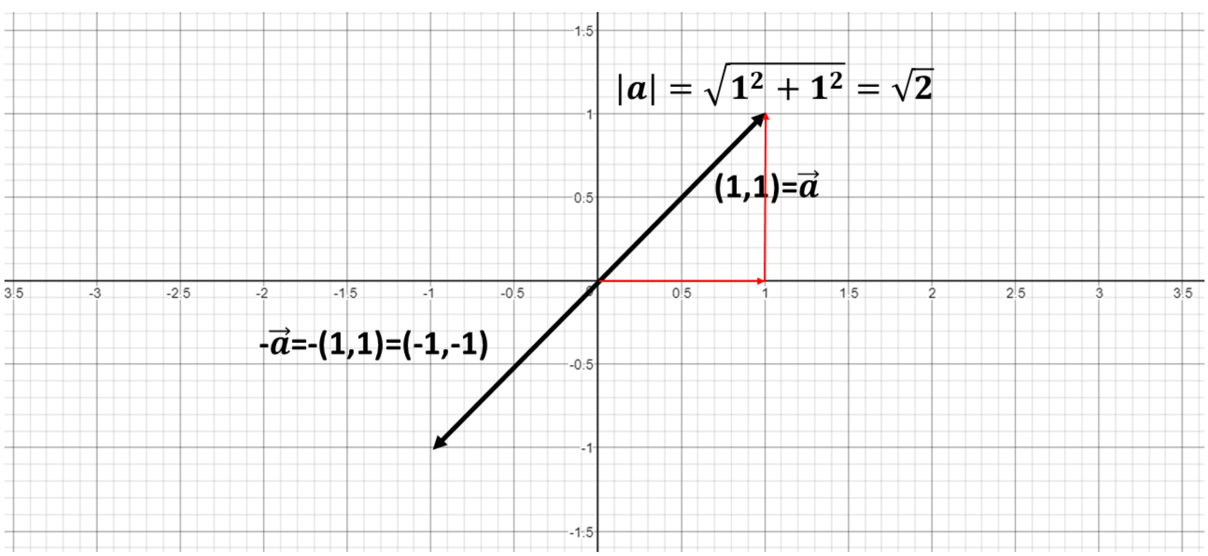

Figure 2.9: Vectors in a Coordinate Plane

48. The number given by the Pythagorean theorem is also known as the **Euclidean**

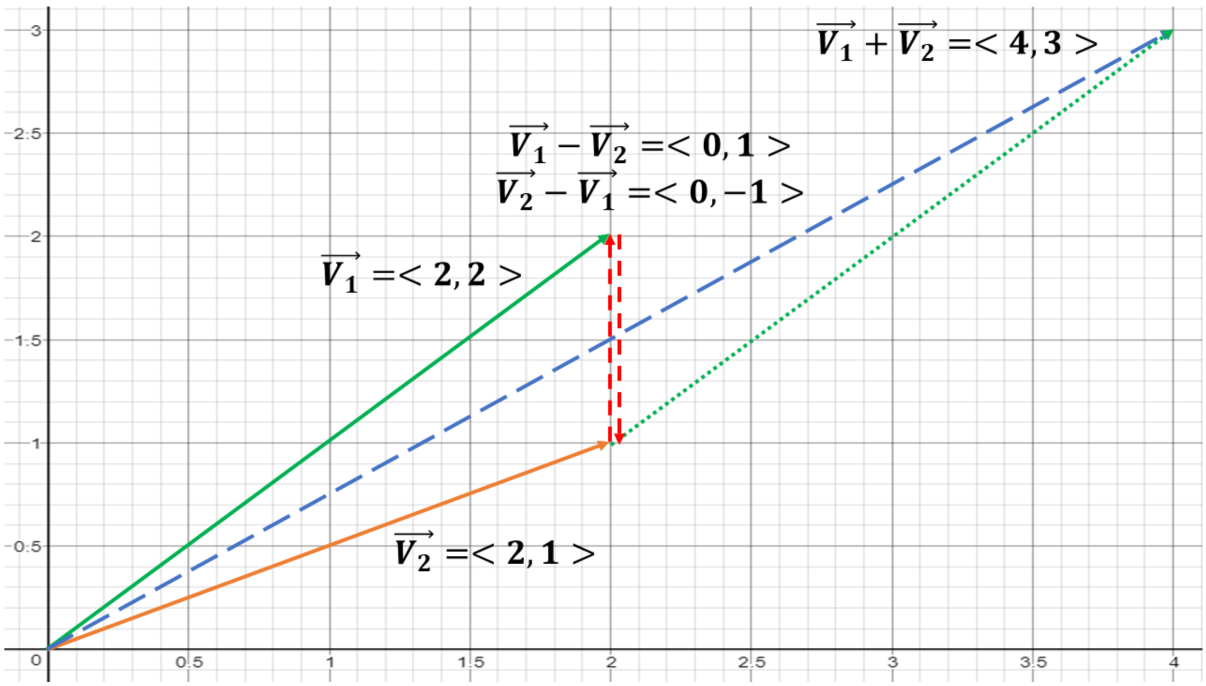

Figure 2.10: Adding and Subtracting Vectors (Coordinate Plane)

distance from two points, $(x, 0)$ and $(0, y)$. Euclidean distance is the length of the shortest possible path through space between two points that could be taken if there were no obstacles in between them.

49. The **negative of a vector** is simply a vector having the same magnitude but of opposite direction as seen on figure 2.9. We can see that the vector $-a$ has the same magnitude but opposite of the direction of vector a. Adding and subtracting vectors are simple in that each component of vector A is added or subtracted to the respective components of vector B. For addition, $C = A + B$ can be written as $(x, y) =< 1, 2 > + < 3, 4 >=< 3, 6 >$. For subtraction, $C = A - B$ can be written as $< x, y >=< 1, 2 > +- < 3, 4 >=< 1, 2 > + < -3, -4 >=< -2, -2 >$. An example is seen on figure 2.10. As we can see, if we add two vectors, $V_1 and V_2$, the resulting vector $< 4, 3 >$ is longer than both vectors if they are both in the same direction. If we subtract the vectors, $V_1 and V_2$ the resulting direction will depend on which vectors are considered as the minuend and subtrahend.

    If we consider a vector in dimension 3, then we will have to add to its components.

Its components are now x, y and z where x is its length, y is its height and z is its width. In general, if we have a vector in dimension n, it is defined with n components.

In this document, we consider the velocity of an object inside a defined virtual space of dimension n.

50. **Velocity** is defined in physics as speed with direction. For example, if an object has a speed of 9 m/s then we can say that the object is simply covering a distance of 9 metric units at each time step but if we state that the object has a velocity of 9 m/s to the right, then we can say that the object is covering a distance of 9 metric units at each time step to the right of its current position. It is important that take note that vectors usually involve two ordered n-tuples that give its original and final positions.

51. An **Array** is a collection of objects, having shared some similar properties, arranged in a particular order. An array is usually contained in rows and columns.

    Arrays are denoted by the syntax ArrayName[Size][Size] wherein every [ ] denotes a **dimension**.

    For example we have the array MyArray[3] it is an array of one-dimension having 3 elements. Take note that the size is sometimes omitted to represent variability. In simple terms, an array is like a series of boxes that contain elements with some similar properties. If we have an array of dimension 2 (MyArray[X][Y]) then we have X rows of Y boxes. A visual representation is shown on figure 2.11. As we can see, the array A[2][5] has two rows and 5 columns. Each element occupies a single box.

    It is common notation to access elements of arrays by its index. Indexing usually starts from 0. In figure 2.11 the red numbers indicate indices of the elements. For example, if we want to access the first element in A from figure 2.11, we say A[0][0]. If we want to access element 'H' in A, we say A[1][2].

    In this paper we will be dealing with arrays that whose element are vectors and coordinates.

52. A **matrix** is an array of numbers.

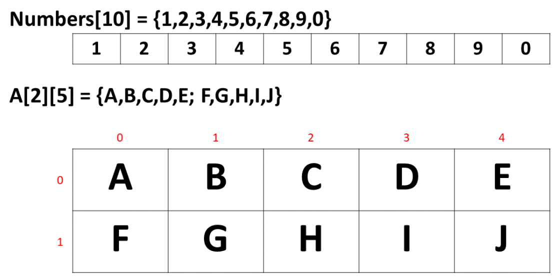

Figure 2.11: Visual examples of arrays

The **dimensions** of a matrix is the number of rows and columns of the matrix in that order. A 'two by three' matrix is an array with two rows and three columns. A 'three by two' matrix is an array with three rows and two columns. To show this, we let $M1$ be a 2×3 matrix and $M2$ be a 3×2 matrix.

$$M1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}, \; M2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

We access the elements of a matrix the same way as we do for arrays. An example is $M1[1][1] = a_{11}$

A matrix whose row and column have the same dimension is called a **square matrix**.

The operations that can be done for matrices are as follows:

[**Matrix Addition**] Adding two matrices means that we add their corresponding elements. We can only add matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then matrix addition $M3 + M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{bmatrix}$$

[**Multiply by a Constant**] Multiplying a constant number c to a matrix M

is done by multiplying the constant to every element of the matrix.

$$c \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} & c \cdot a_{13} & \dots & c \cdot a_{1m} \\ c \cdot a_{21} & c \cdot a_{22} & c \cdot a_{23} & \dots & c \cdot a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c \cdot a_{n1} & c \cdot a_{n2} & c \cdot a_{n3} & \dots & c \cdot a_{nm} \end{bmatrix}$$

[**Negative of a Matrix**] The negative of a matrix is just the matrix multiplied to the constant $c = -1$ hence, all elements are multiplied to -1.

$$-\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} = \begin{bmatrix} -1 \cdot a_{11} & -1 \cdot a_{12} & -1 \cdot a_{13} & \dots & -1 \cdot a_{1m} \\ -1 \cdot a_{21} & -1 \cdot a_{22} & -1 \cdot a_{23} & \dots & -1 \cdot a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 \cdot a_{n1} & -1 \cdot a_{n2} & -1 \cdot a_{n3} & \dots & -1 \cdot a_{nm} \end{bmatrix}$$

[**Matrix Subtraction**] Matrix subtraction is just the addition of two matrices where the addend is negative. Note that here, we can only subtract matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then matrix subtraction $M3 - M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} & \dots & a_{1m} - b_{1m} \\ a_{21} - b_{21} & a_{22} - b_{22} & \dots & a_{2m} - b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} - b_{n1} & a_{n2} - b_{n2} & \dots & a_{nm} - b_{nm} \end{bmatrix}$$

[**Hadarmard Product**] The Hadamard Product is a **component/element-wise multiplication** where each element is multiplied to the corresponding element of the other matrix. Note that here, we can only multiply matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then their hadamard product is given by $M3 \circ M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & \dots & a_{1m} \cdot b_{1m} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & \dots & a_{2m} \cdot b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} \cdot b_{n1} & a_{n2} \cdot b_{n2} & \dots & a_{nm} \cdot b_{nm} \end{bmatrix}$$

[**Matrix Multiplication**] Matrix multiplication is not the Hadamard product. Matrix multiplication involves the sum of products. If A is an $n \times m$ matrix and B is an $m \times p$ matrix, their matrix product AB is an $n \times p$ matrix, in which the m elements across a row of A are multiplied with the m elements down a column of B, the resulting elements are then summed to produce an entry of AB. Let two matrices A and B be matrices of the sizes $n \times m$ and $m \times p$ respectively, then their matrix product is given by $A \times B$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{np} \end{bmatrix}$$

such that

$$c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \cdots + a_{im} \cdot b_{mj} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}, \forall i = 1, 2, \dots, n \text{ and } j = 1, 2, \dots, p$$

An example is

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} z_{11} \end{bmatrix}$$

$$z_{11} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

53. The **transpose** of a matrix (denoted as M^T) is a matrix where the rows and columns are swapped. That is

$$M3^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \dots & a_{mn} \end{bmatrix}$$

An example is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \ A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

54. A matrix is said to be **symmetric** if and only if the matrix M is equal to it's transpose M^T. Given by $M = M^T$. An example is

$$O = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} = O^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}, \therefore O \text{ is symmertric}$$

55. A **graph** G is a mathematical object composed of two sets, a finite set V called the **vertices** and another set E whose elements are pairs of vertices called **edges**, expressed as $G = (V, E)$.

56. A **vertex**, also called a **node**, is the fundamental unit needed to construct graphs. They are visually represented as points in some space S having N dimensions. In this document, they are used to represent real world objects. Later, we will assign numbers to these points to achieve discreteness, (to know what they are and what they are not).

57. **Edges** are visually seen as lines that connect vertices, they show that those vertices are related in some way. Edges usually connect two vertices, they represent and show that there exists a relationship between these vertices. If there are no edges that connect a pair of vertices, then it an be said that there is no direct relationship between those edges.

58. If two vertices $u, v \in V$ are connected by some edge $(u, v) \in E$, and if the edge $(v, u) \in E$ is the same edge, then we say that vertices u and v are connected by the **undirected edge** (u, v) (or (v, u)).

59. We also say that the vertices $u, v \in V$ are **adjacent** because an undirected edge connects them.

60. A graph G is called an **undirected graph** if and only if it is made up of undirected edges.

61. However, if edge $(u, v) \in E$, and $(v, u) \in E$ are not the same edges, then we say that (u, v) is a **directed edge** from vertex u (called the edge's 'tail') to vertex v (called the edge's 'head').

62. If edge $(u, v) \in E$ but $(v, u) \notin E$, then we say that vertex u is **adjacent** to vertex v but vertex v is not adjacent to vertex u.

63. A graph G is called a **directed graph** if and only if it is made up of directed edges.

64. A graph with which every pair of vertices $u, v \in V$ is connected by an edge $(u, v) \in E$ is called a **complete graph**, denoted as $K_{|V|}$. That is, there exists an edge (u, v) in set E for any pair of u and v in set V (expressed as $\exists(u, v) \in E \; \forall u, v \in V$).

65. A graph is said to be a **weighted graph** if numbers are assigned to it's edges. These numbers are called **weights** or **costs**.

66. A **path** from vertex u to vertex v of a graph is defined as a sequence of adjacent vertices (connected by edges) that start from u and end with v.

67. If all vertices of a path are distinct, then the path is said to be **simple**.

68. The **length** of a path is the total number of edges in the path.

69. A **directed path** is a sequence of vertices in which every consecutive pair of the vertices u and v is connected by a directed edge from u to v.

70. A graph is said to be **connected** if for every pair of vertices u and v in set V, there exists a path from u to v.

71. A **cycle** is a path of positive length (at least one edge) that starts and ends at the same vertex and does not traverse the same edge more than once.

72. A graph with no cycles is said to be **acyclic**.

73. An **adjacency matrix** is a square matrix that shows the relationships of vertices in a graph. Each dimension in the matrix is assigned a vertex. The elements of the matrix is from the set $0, 1$. The element $M[u][v] = 1$ if there is an edge that connects vertices u and v, otherwise, is it 0. The unweighted graph in figure 2.12

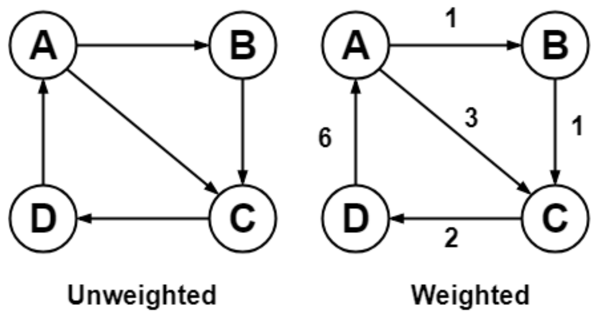

Figure 2.12: Sample Graph

has the adjacency matrix:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

If the graph has weights then we replace the 1's with their respective weights. The adjacency matrix of the weighted graph in figure 2.12 is:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 6 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Note that for an undirected graph, the adjacency list is symmetric.

74. The **shortest path problem** is the problem of finding a path between two vertices (or nodes) u and v in a graph G such that (a) if G is unweighted, the total length of the path is minimized; (b) if G is weighted, the sum of the weights of the edges in the path is minimized.

    The well known algorithms used to solve the shortest path problem are as follows:

    (a) **Dijkstra's Algorithm** which solves the shortest path problem with non-negative weights. It is an algorithm for solving the single-source shortest

path, which means that it solves the shortest path from any node $u \in V$ to any other node $v \in V$.
The dijkstra's algorithm uses a priority queue.

A **queue** is a list where the elements are inserted at one end and are removed at the other.

A **priority queue** is a queue wherein each element is associated with a value which dictates whether or not that element is highly likely to be selected/removed from the queue. An element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.
The dijkstra's algorithm is:

i. Select a source vertex s from the set of vertices V

ii. Create an empty priority queue Q

iii. For each vertex v in the Graph, do the following
  - Set the distance from the source s to vertex v as infinity (∞)
  - Set the optimal path from the source s to node v as empty
  - Add vertex v to the priority queue Q

iv. Set the distance of vertex s from itself as 0

v. While Q is not empty, do the following:
  - Select vertex u from the priority queue Q with the minimum distance
  - Remove u from the queue
  - For each vertex w, still in the queue, adjacent to u, do the following:
    - Compute the path from the source vertex s to the node w that passes through u before it reaches w
    - If the newly computed path is shorter than the current one,

      Update the distance from the source vertex s to vertex w

      Add vertex u to the path of w

The flowchart of the algorithm is seen on figure 2.13.

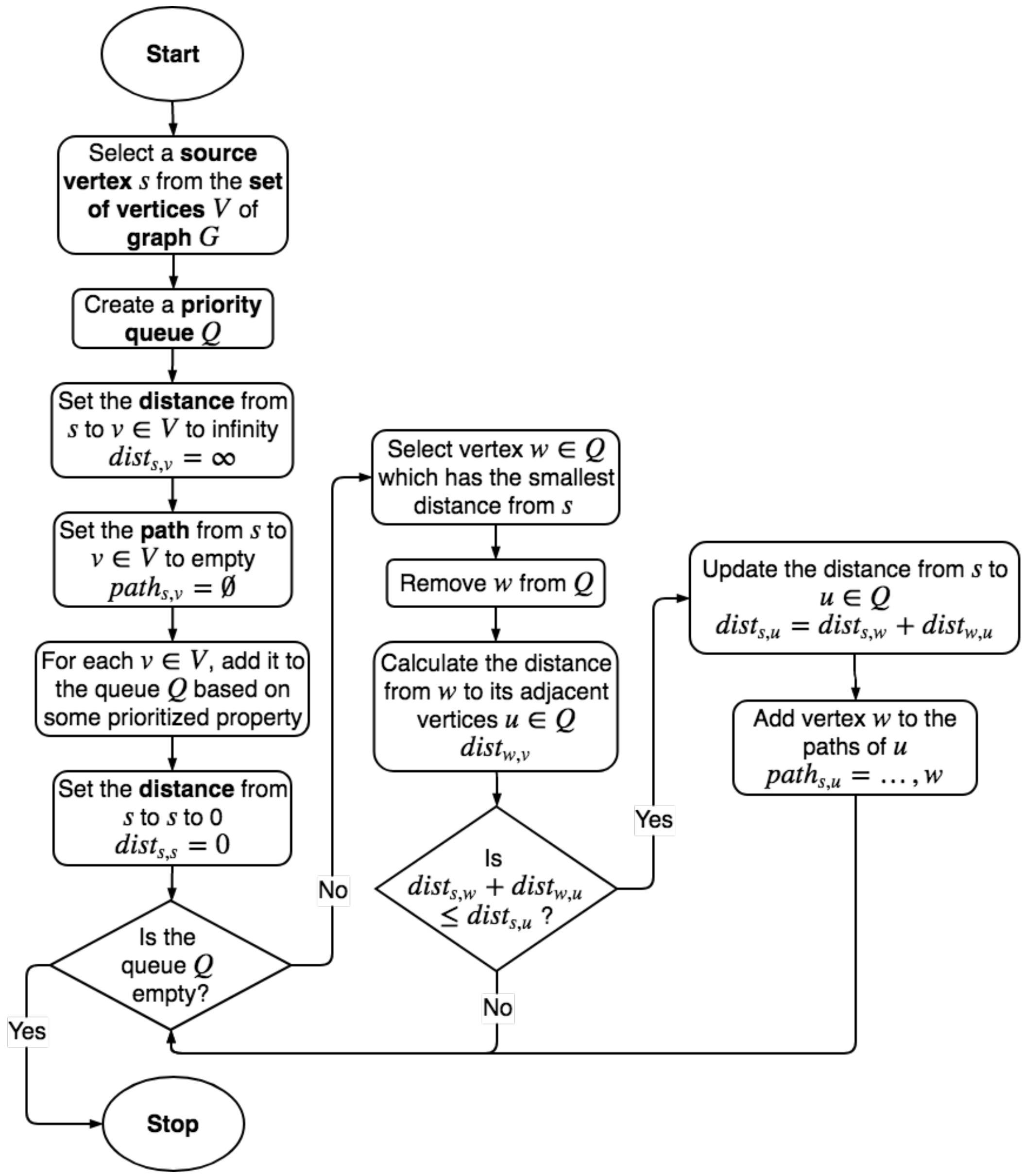

Figure 2.13: Flowchart of the Dijkstra Algorithm

(b) **Floyd-Warshall Algorithm** which solves the shortest path for any two node u and v in V. The floyd-warshall algorithm starts off with the adjacency matrix of the graph G. All non-existent edges have the value of infinity ∞. The algorithm takes advantage of the transitivity in order to replace the infinite values. Transitivity is the relation wherein if a property holds between the first and the second and also holds between the second and the third, then it follows that this property also hold between the first and the third. It can be simplified as "if one can go from a to b and from b to c then one can go from a to c by passing through b."

The Floyd-Warshall algorithm is as follows:

i. Let $dist$ be a matrix of size $|V| \times |V|$ whose values are ∞

ii. For each edge, $(u, v) \in E$, set $dist_{u,v}$ as the weight of the edge (u, v).

iii. For each vertex $v \in V$, set the distance to itself as 0. $dist_{u,v} = 0$

iv. For each vertex $w \in V$, do the following:

- For each pair of vertices $u, v \in V$ do the following:
  - Check if the distance from u to v is greater than the distance from u to w and w to v. That is, check if $dist_{u,v} > dist_{u,w} + dist_{w,v}$
  - If that is true, set $dist_{u,v} = dist_{u,w} + dist_{w,v}$

The flowchart of the floy-warshall algorithm is seen on figure 2.14. An example is shown on figure 2.15.

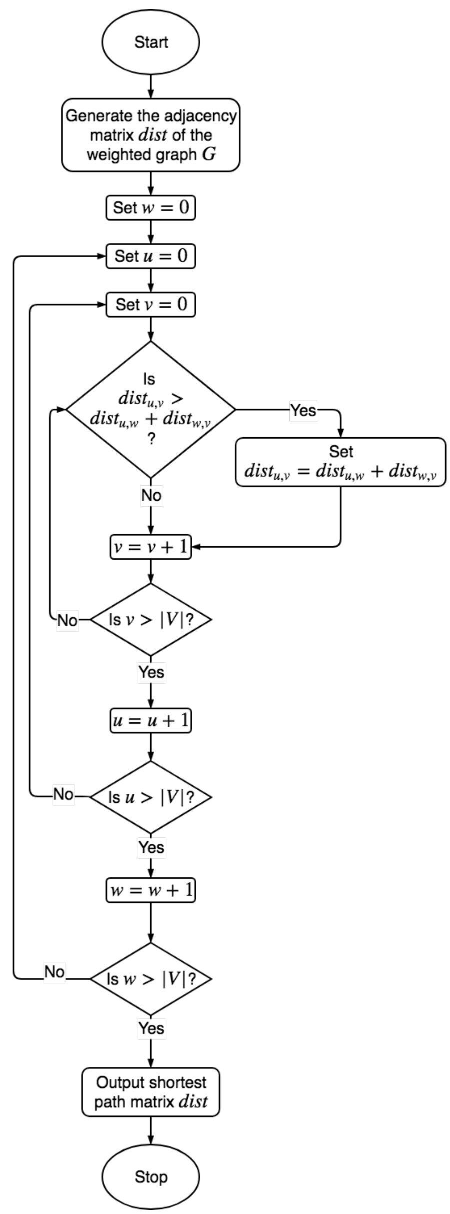

Figure 2.14: Flowchart of Floyd-Warshall Algorithm

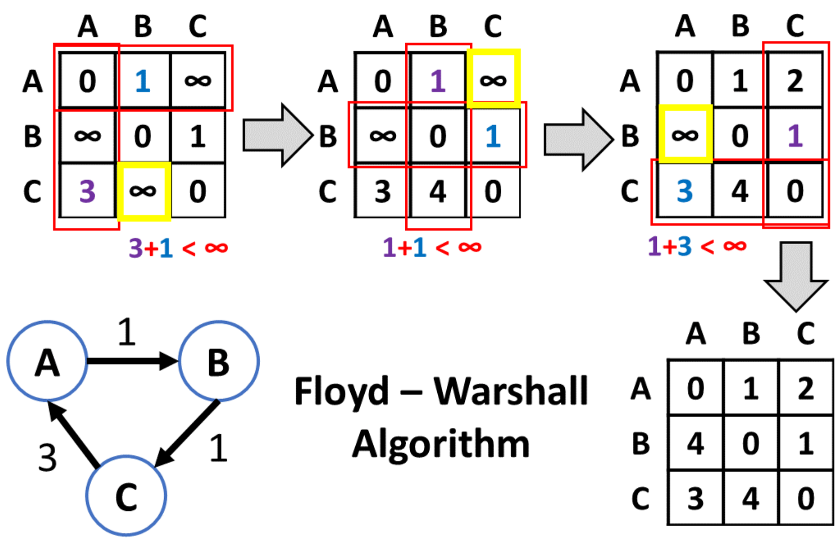

Figure 2.15: Floyd-Warshall Algorithm Example

An application of this algorithm involves finding a sequence of road segments that take a vehicle from a source to a destination using a graph that represents a road network. In this representation, we can let vertices be the source, destination, intersections, land marks, etc. whatever objects that can help split the entire road systems into road segments. We then let edges be the road segments between any two vertices. We assign the costs/weights to the edges based on some information that can help us understand and/or distinguish edges that are favorable to traverse. These costs may be quantified as actual distances, cost of fuel, average amount of travel time, traffic gradient, risks involved (bridge instabilities, accident proneness, etc.) and much more depending on the realism of the model and/or data availability. The model for the amount of cost can be as simple as minimizing the amount of distance traveled or as complex as maximizing the total amount of money gained after subtracting the total money expended on fuel (affected by both distance and time), car maintenance, driver salary, etc.

75. The **Traveling Salesman Problem (TSP)** is a problem involving generating a **Hamiltonian Cycle** from a graph G.

A hamiltonian path is a path in a graph which contains each vertex of the graph

exactly once. A hamiltonian cycle is a hamiltonian path that starts and ends at the same vertex.
The problem description are as follows:

(a) A salesman needs to visit every city (represented by vertices)

(b) He/she does not care about the order of visiting each city. As long as he/she visits each one.

(c) He/she must start and finish at the same city

(d) Each city is connected to other close by cities, or nodes, by airplanes, or by road or railway. Hence, each of the connections between the cities has one or more weights (or the cost) attached depending on the availability of transportation means.

(e) The cost describes how "expensive" it is to traverse this edge on the graph, and may be given, for example, by the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required to complete the traversal.

(f) The salesman wants to keep both the travel costs, as well as the distance he travels to a minimum.

The aim is to generate a path or a sequence of nodes that lets the salesman pass through all cities at most once before returning to starting city and spends the minimum amount of travel expenses and distance.
The problem is mostly concerned about generating the best arrangement of N cities among $(N-1)!$ permutations. As we can see, the amount of permutations rapidly increases as the number of cities is increased. $N-1$ because we always start with the same given city.

76. The **Vehicle Routing Problem** is a generalization of the Traveling Salesman Problem. VRP is a problem that involves generating the best set of routes for a fleet of vehicles to service all customers in a graph. Here, there are more 'salesmen' (changed into 'vehicles' for formalities when we consider modern delivery services). VRP is concerned with delivering or collecting 'goods' to and/or from customers using a number of vehicles.

A **route** is a hamiltonian cycle which starts and ends at a depot.

A **depot** is where vehicles are stored or parked when they are not in use.

The usual way customers and road networks are set-up is to let vertices represent the depot and customers and let the edges represent road segments that connect the vertices. Another way is to represent clusters or customers as an edge, and let the vertices serve as road intersections. This is simple but it is too simple that it does not capture individuality of customers. Hence, the former is commonly used since most adaptable models are complex.
VRP is defined on a complete undirected graph $G = (V, E)$. The set of vertices $V = 0, 1, 2, \ldots, n$ where each vertex $u \in V - \{0\}$ represents a customer having a nonnegative demand q_u. The demand is usually the amount of goods (in some quantity) to be delivered or collected by the vehicle. The amount of goods can be measured in mass, weight, quantity, volume, bulk, etc. Vertex 0 is usually designated as the depot. Each edge $e \in E = (u, v)|u, v \in V$ is associated with a travel cost c_e or $c_{u,v}$. Travel cost may be in terms of distance (actual, euclidean, circular, manhattan, chessboard), time (travel time, time waiting in traffic), fuel cost (convert distance and time into amount of fuel and convert that number into how much money fuel costs), monetary cost (adding up expenses, salaries, penalties) etc. There are a total of k available vehicles in the depot. The vehicles are assumed to be homogeneous and all have the same carrying capacity Q. Carrying capacity refers to the maximum amount of goods that can be carried by a vehicle at any phase or time during it's traversal of the route. The task is to develop k routes whose total travel cost is minimized such that

- Each customer is visited exactly once by a route
- Each route starts and ends at the depot
- The total demand of customers served by a route does not exceed the vehicle capacity Q
- The length of the route does not exceed a preset limit L

The last item ensures that all the drivers have the same workload.

If we consider a directed graph, then we need only to change the edges and must produce directed cycles.

77. **Waste** is defined as materials that are unwanted or unusable. This refers to matter which are discarded after primary use or is deemed defective, or worthless.

78. There are various types of waste but our focus is one **municipal solid waste** or what we call household trash. This is the type of waste that is produced daily at households and communities. The term 'municipal' comes from the fact that it is the duty of the municipality to collect and manage these kinds of waste. A list of what is considered as municipal solid waste is as follows:

    - **Biodegradable waste** produced from food, cooking, paper etc.
    - **Recyclable materials** such as glass, bottles, jars, clothes, fabrics, rubber etc.
    - **Inert waste** such as dirt, rocks, debris
    - **Electrical and electronic waste** such as appliances, light bulbs, mobile phones, television sets
    - **Composite waste** such as toys, tetra packs, clothing
    - **Hazardous waste** such as paints, batteries, aerosol sprays, and fertilizers
    - **Toxic waste** such as pesticides, herbicides and fungicides
    - **Biomedical waste** such as expired pharmaceutical drugs, used medical equipment

79. **Waste collection** includes gathering, transportation, and delivery for disposal of solid waste and recyclable materials. Waste collection involves vehicles that collect and transport the waste from communities to facilities that receive, sort and process the waste. Processing the received waste may be in the form of incineration, rapid degradation, segregation, resource recovery, energy recover, etc.

80. **Residual waste** is the type of solid waste that is neither recyclable nor reusable.

81. An **Eco-Waste Recovery Services-Material Recovery Facility** is where final sorting of waste is done. Once sorted, the garbage is then moved to designated

areas for recycling, recovery, reuse, re-purposing, composting, etc. This facility reduces the amount of residual waste and also corrects any mis-segregated matter.

82. A **Geographic Information System (GIS)** is a collection of computer software, and data used to view, manage, analyze, and transform geographical information. A GIS provides a framework for gathering and organizing spatial data and related information such as temporal, visual, demographic, economic, etc. Out of the data available, it is able to produce analyses, maps, patterns, predictions, assessments and other forms of usable information. It can create fast and logical decisions, produce and display maps, graphs, charts and perform a vast quantity of calculations. An example is Google Maps which offers satellite imagery, street maps, 360 deg panoramic views of streets (Street View), real-time traffic conditions (Google Traffic), and route planning for traveling by foot, car, bicycle (in beta), or public transportation. These kinds of information was produced through available data and some algorithms which processes the data for generating visuals and graphics, route creation, land mark associations etc. Data that is stored in a database is placed in several layers of maps and graphs that have common properties. These layers can come in the form of roadways, vegetation patterns, layout of buildings and structures, traffic information, physical layout of the environment, temperature and pressure maps, radiation maps, demographics, environmental compositions, sets of images and videos, etc. Informally, a physical map is itself a GIS. Within it, is information which can be read and analyzed to produce observations, inferences, hypotheses, predictions, plans, patterns, etc. Basically, it is anything that can tell you something about a place. It has been used in businesses for needs assessments, sales predictions, discovering patterns of customer interests, discovering trends in purchases and demand.

83. **Deterministic** means that the next procedure/step is known without having any other choice. There is no randomness involved.

84. **Non-deterministic** means that there are multiple available decisions that can be done at a certain circumstance. This helps examine the ability to make decisions based on the statements.

85. **Decision problems** is any yes-or-no question that involves an infinite set of inputs. These inputs are logical objects, be it numbers, graphs, strings, or sets. The input is broken down to its properties and based on those properties, the question is thrown an affirmation or negation. For example, "is 4 an element of $\mathbb{Z}$? Well we can compare all numbers in $\mathbb{Z}$ to 4 and this will of course be true, hence the answer returned is 'yes' 4 is an element of $\mathbb{Z}$.

86. Nondeterminstic Polynomial time **NP** is a set of problems with the same resource-based complexity used to describe certain types of decision problems. NP is the set of all decision problems for which the instances where the answer is "yes" are efficiently verifiable through deterministic computations that can be performed in polynomial time. A problem belongs to the $NP - hard$ or the set of the "hardest" NP problems if there is no known polynomial time algorithm that can provide an optimal solution.

87. A **constrained optimization problem** is a problem that is bounded by some limiting factors. According to Garg[14], Constrained optimization problems are defined as:

$$\text{Minimize } F(x)$$

that is subjected to p equality constraints,

$$h_k(x) = 0; \qquad k = 1, 2, \dots, p$$

and q inequality constraints,

$$g_j(x) \leq 0; \qquad j = 1, 2, \dots, q$$

where each set of decision variables x is in D dimensions such that $x = [x_1, x_2, x_3, \dots, x_D]^T$. Each element of x are bounded as

$$l_i \leq x_i \leq u_i; \qquad i = 1, 2, \dots, D$$

where l_i and u_i are the minimum and maximum permissible values of each element x_i.

88. **Feasible solutions** are solutions that do not violate any constraint imposed in a constraint optimization problem. The solution space S is also called the 'feasible space' since it composes of all solutions that are within the limitations of the problem.

89. **Infeasible solutions** are solutions that do violate any constraint imposed in a constraint optimization problem.

Chapter 3

Review of Related Literature

As mentioned in the previous chapter, waste collection problems have been solved through the use of the vehicle routing problem. We take a look at some of the studies conducted in solving vehicle routing problems and modeling waste collection into a vehicle routing problem. We then move on to discus the basic PSO and GA algorithms. Finally, we discuss the hybrid PSO-GA approach of Harish Garg.

3.1 Vehicle Routing Problem

In 2007, Cordeau et.al.[6] compiled and defined general models of the vehicle routing problem and its extensions (i.e. capacitated, time windows, etc.). They also collated and cited several approaches done by researchers over the years to tackle the VRP and it's extensions. They give a brief description of the process of how each algorithm solves the problems presented.

As stated, Dantzig and Ramser[7] were the first to state the vehicle routing problem. Their paper focused on routing a fleet of gasoline-powered delivery trucks that deliver fuel from a 'bulk terminal' to a large number of service stations. The bulk terminal is a facility that stores petroleum products. Service stations are facilities where gasoline-powered vehicles refill their tanks, usually, vehicle repair is also included in the services offered. Dantzig and Ramser stated that the traveling salesman problem, at its core, is merely concerned with determining the shortest possible route which passes through each of the n given cities exactly once. Assuming that for each pair of cities, there exists some link that directly connects them to and for, then therefore the total number of distinct routes through n cities is given by $\frac{1}{2}n!$. This is because the sequence at which cities are visited is the same as in the revere order since the salesman returns to the same city.

The generalization of the TSP is made by adding more conditions to the problem. They basically thought about the possible outcome when there was a limit to the number of cities that the salesman can visit before returning to the origin city. The salesman would have to take an increasing number of shorter trips every time the maximum number of cities that can be visited is reduced. The context was changed into a delivery truck that transports fuel from the bulk terminal to some n service stations. Hence, a limit was introduced by giving each service station i $(i \in 1, 2, \ldots, n)$ a quantity q_i equivalent to the amount to be delivered to that service station. The vehicle is imposed to only have the ability to carry a total amount of C fuel every time it is deployed from the bulk terminal. It was set that $C > q_i$ for any service station $i \in 1, 2, \ldots, n$. Hence, the number of service stations that can be serviced by the vehicle in a single route is determined by the demand q_i of each service station $i \in 1, 2, \ldots, N$. The vehicle is now forced to make multiple deliveries when the sum of all q_i's is greater than the vehicle's capacity C. The main goal was still the same, minimize total travel distance covered by the vehicle. Another interpretation made from imposing the limit is that instead of a single vehicle taking on multiple trips, each trip is assigned a different vehicle hence, there are multiple vehicles with the same capacity C that deliver fuel to the same set of service stations. Dantzig and Ramser solved the truck dispatching problem using integer linear programming. This is a method where the solution is obtained by using the linear relationships of the mathematical models in terms of their graphs. A limited space is usually produced when the equations are plotted in a single graph, this space is called the solution space. The best solutions are then identified using the points near or at the boundaries of this space.

In 1987, Solomon[23] proposed some methods of constructing routes in order to solve the Vehicle Routing Problem. He tested them on some problems sets he created called the "Solomon Benchmark Problems" which are used for benchmarking solution methods used to solve Vehicle Routing Problems with Time Windows. A time window is a span of time that dictates when a customer is ready to be served by a vehicle. Time windows consists of the earliest and latest possible time that a customer can be served. Outside of a customer's time windows, no vehicle is allowed to service that customer. The first route construction algorithm is called the 'savings' heuristics which starts out with each customer having dedicated routes. This means that each customer is exclusively serviced in a single vehicle trip. The algorithm then tries to combine the best pair of routes until the minimum amount of routes are produced. The best pair of routes is decided by some savings equation which gives the amount of cost saved if two routes are combined rather than separate. The best pair gives the most amount of cost saved. The next heuristic is a greedy approach. This means that in any situation, the best possible choice is selected without thinking of future consequences. The time-oriented nearest-neighbor heuristic starts by selecting the 'closest' node from the depot and attaching it to the current route. 'Closest' means that the node nearest to the depot in terms of travel distance or time. The process is repeated until the vehicle's schedule is full however, we select the closest node from the last added node instead. The algorithm proceeds to create the next route if there are still un-routed nodes. The next heuristic introduced is the insertion heuristic which constructs routes sequentially. The route starts as two depot nodes. Each customer node is then inserted between two consecutive nodes in the route. The best location for insertion is determined by some function that shows how efficient the route becomes after insertion. There are three proposed ways of evaluating the efficiency of the routes each called $I1$, $I2$, and $I3$ respectively. $I1$ evaluates the route by distance and start of service time; $I2$ evaluates the route by total distance and total time; $I3$ evaluates the route by a combination of total distance, total travel time, and total time vehicles are late. When the node is inserted, the nodes succeeding it in the route are *pushed forward*, meaning that servicing these nodes are adjusted based on how much time and distance is used to accommodate the inserted node. The last heuristic discussed is the time-oriented sweep heuristic which groups customers into clusters and assigns each cluster to a vehicle. The route construction and scheduling is then done for each cluster of nodes associated to a vehicle. The results show that among the proposed heuristics, the insertion algorithm (specifically the $I1$) proved to be the most effective in solving the benchmark test cases because it focuses more on correct node sequencing rather than grouping customers in a vehicle's route.

In 2000, Son[33] utilized a Chaotic Particle Swarm Optimization (CPSO) algorithm to generate routes and schedules of the different waste collection vehicles at Danang City Vietnam. PSO is discussed later in this chapter. The CPSO obtained data on the roads and waste collection facilities from a Geographic Information System (GIS) that simulates a continuous environment from a model of the road networks and waste collection system of Danag City. The information used in the simulation of the GIS are a collection of real data obtained through a span of time. From this data, the average amount of waste collected at an area is known and is then simulated to vary based on the average amount. Traffic and other variables taken into consideration are also simulated the same way. There are three different kinds of vehicles available, namely, tricycles, hook-lifts and forklifts which take up different roles in the waste collection system. The objective in this case was to create a schedule that maximizes the amount of garbage collected in the simulation.

In 2005, Nuortio et.al.[26] improved the inflexible and inefficient waste collection scheduling and routing in Eastern Finland by creating a GIS model that is made based on the available road network and waste collection data. They employed a hybrid insertion heuristic for generating the initial population. A guided variable neighborhood thresholding meta heuristic was then used for improving the initial routes. This heuristic is based on three principles, (1) guided local search, which performs a search on the search space S with the intent of finding the local optima. The decision of selecting which part of the search space to explore is based on a deciding factor that 'guides' the search. (2) variable neighborhood search which explores a particular local search space while executing the same local searching approach on adjacent neighborhoods (local search spaces) and switches the current local search space being explored with the neighborhood that shows a better or promising solution. (3) Threshold accepting is a method of evaluating the solutions found and judging whether or not the solution is a good enough approximation of the best solution. This is done for when obtaining the best solution becomes inefficient in terms or resources so instead, it is better to settle for a close approximate. The result of their experimentation showed that the schedule produced by the heuristic significantly reduced traveling distance of vehicles.

In 2012, Burhkal et.al.[2] set-up a model for waste collection vehicle routing problem with time windows (WCVRPTW) with lunch breaks based on two test cases, namely that of the (1) Waste Management Inc. which is responsible for waste collection in parts of Northern America and (2) the Henrik Tofteng Company responsible for handling waste collection at Denmark. These two cases have different policies for lunch break hours, limits on the number of customers served per route, and total amount collected at each route. They provided both cases with solutions using an adaptive large neighborhood search heuristic. Neighborhood search is a technique that tries to find good or near-optimal solutions to a combinatorial optimization problem by repeated transformation of a current solution into different solutions in its 'neighborhood'. The neighborhood of a solution is a set of similar solutions obtained by relatively simple modifications to the original solution (i.e. swapping two nodes in the route). For a large-scale neighborhood search, the neighborhood produced from a solution is relatively numerous in count since there are more variables taken into consideration. The 'adoptive' part stems from the fact that the algorithm tries to improve the solution by adjusting the neighborhood produced using the current known solutions at each iteration or time-step. They found that the algorithm produced considerable improved routes from those being used by the two companies.

In 2015, Akhtar, Hannan and Basri[1] proposed a method of solving Waste Collection Vehicle Routing Problem by node clustering in order to simplify the problem. They distributed customers into bins and modeled a traveling salesman problem for each cluster/bin. They then applied the Particle Swarm Optimization algorithm to find optimal routes for each TSP. This method is based on the notion of divide-and-conquer however, note that the clustering method used is significant since it determines which nodes go to which route. In their approach, they used smart bin technology which sends information about each bin specifically the location and current amount.

Masrom et.al.[24] developed a hybrid PSO by incorporating the mutation mechanism of GA. Each particle is made up of $n + 2m$ components where n is the number of customers and m is the number of vehicles. The initial population is made through assigning real numbers to each component. The n components associated with customers are distributed to m vehicles using the real numbers. The customers are assigned to the vehicle whose associated real number is closest to the value of the customers' real number. PSO is followed in each iteration and Mutation only occurs when the population's total health is low. A 'healthy' particle is one that changes it's personal best at each iteration. A population with low total health is a population where the majority of particles have not changed their personal best positions therefore we can say that the population might have fallen into a local optima and has stagnated. Mutation maintains that the population keeps moving even if only at a small distance. Both PSO and GA algorithms are discussed later in this chapter.

Lu et.al [22] also developed a hybrid PSO algorithm however this time, the crossover mechanism of GA was used. Each particle position has $n+l-1$ components where n is the number of customers and l is the number of vehicles. Integers are assigned to each component which allows for the creation of the initial population. The position components are arranged using the integers assigned hence, the sequence of visitation of each node is established. Each vehicle's route is given by the sequence of customers between two vehicle components. This is visualized as follows, If we have 3 customers and 2 vehicles, each particle position in the population will have 4 components. We let node 0 to be the depot and nodes 1 through 3 as the collection sites. Given

$$\begin{array}{cc} \text{Nodes} & \begin{matrix} \scriptstyle 1 & \scriptstyle 2 & \scriptstyle 3 & \scriptstyle 0 \end{matrix} \\ \text{Particle} & \begin{bmatrix} 3 & 4 & 1 & 2 \end{bmatrix} \end{array}$$

The route is given as $0 \leftarrow 3 \leftarrow 0 \leftarrow 1 \leftarrow 2 \leftarrow 0$. Notice that both vehicles are used and they have different routes. Crossover is done using two particles, a section of the position vector is selected in each particle and removed from that particle. The section is then placed at the beginning of the other particle's position vector. This is done for both particles, hence two new sequences are created which replaces the old ones. We give an example. Given two particle positions [3412] and [2143]. We take the last two sections of each array and place them at the beginning of the other array. Hence we have the two new particles [3421] and 4312. Their results showed that the hyrbid PSO outperformed the basic PSO and basic GA algorithms.

In 1999, Tung and Pinnoi[36] conducted a case study wherein they investigated the refuse collection of a public company (URENCO) in five urban district of Hanoi, Veitnam. The aimed to improve its daily operation, particularly its their vehicle routes and schedule. The collection of waste involved two types of vehicles, motorized vehicles and manually pushed handcarts. The handcarts were used to manually gather refuse from each household or industrial unit. The refuse was then transported to gather sites where the motorized vehicles collect. Each gather site had a set schedule based upon the arrival of vehicles and the time it took for handcarts to deliver the refuse to the site. The motorized vehicles, after having been filled, transport the refuse to a landfill and return to servicing gather sites. The workers were separated into three shifts; morning, afternoon and night. They implemented both route construction and improvement methods. Route construction was done with the I1 insertion heuristic of Solomon. Route improvement manipulates the route constructed by the insertion heuristic in order to obtain better routes. Two route improvement methods were used, either method is invoked in an alternating or random pattern. The Or-opt exchange modification tries to improve a route by removing up-to-three adjacent nodes and reinserts them at different locations within the same route. The 2-opt operation on the other hand removes two edges, one from two selected routes and replaces them with edges wherein the first selected route is connected to the detached segment of the second route and the second route is connected to the detached segment of the first route.

3.2 PSO

Particle Swarm Optimization (PSO) is an optimization algorithm based on a simplified avian social model. PSO was proposed by Kennedy and Eberhart on 1995.[10][9] The PSO algorithm is seen on algorithm 1. PSO was discovered from the attempts to simulate bird flocking and fish schooling. It has been used to solve a wide array of optimization problems ranging from simple root finding to complex engineering optimization problems. The flowchart for the algorithm is shown on figure 3.1.

The original algorithm is quite simple. The population is initialized by randomly obtaining some particles within the search space and generating random velocities that are paired to each particle. There are N particles in the population. Each particle's position ($x_{i,d}$) and velocity ($v_{i,d}$) are composed of D numbers where D is the dimension of the search space S and $i \in (1, 2, 3, \ldots, N)$. We take note that each dimension of the search space is usually bounded or are in intervals $[a_j, b_j]$, $j \in (1, 2, 3, \ldots, d)$. a_j is the lowest number that each x_{ij} can be while b_j is the highest number that each $x_{i,j}$ can be.

Each particle's personal best value and location are recorded as *pbest* and $pbest_{id}$. In

Algorithm 1: PSO Algorithm

Input : Parameters:
Population Size N, Maximum Iterations M, Problem Dimension D, Cognitive Bias c_1 and Social Bias c_2, Boundary Conditions $[l, u]$ of each component, Velocity Boundaries $vmin$ and $vmax$

Output: Optimal Solution $x_{b,d}$

for $i = 1 : N$ **do**
  `//` $d \in 1, 2, \ldots, D$
  Initialize the position of particle i with a uniformly distributed random vector of d dimensions: $x_{i,d} \sim \bigcup(l, u)$
  Initialize the velocity of particle i with a uniformly distributed random vector of d dimensions: $v_{i,d} \sim \bigcup(vmin, vmax)$
end
$j \leftarrow 1$
while $j <= M$ **do**
  Evaluate the fitness function values $F(x_{i,d})$ of each particle $x_{i,d}$, $i = 1, 2, \ldots, N$
  `// Initialize or change the particle` $x_{i,d}$`'s personal best location` $pbest_{i,d}$
  if $j == 1$ *or* $pbest > F(x_{i,d}))$ **then**
    $pbest(x_{i,d}) \leftarrow x_{i,d}$
  end
  `// Initialize or change the` j^{th} `population's global best location` $pbest_{g,d}$`,` g `is the index of the prevoius population's best particle.`
  `//` b `is the index of the of the current population's best particle`
  if $j == 1$ *or* $gbest > F(x_{b,d}))$ **then**
    $pbest_{g,d} \leftarrow x_{b,d}$
  end
  Update the velocities and positions of the population according to the equation:

$$v_{i,d} = v_{i,d} + c_1 \cdot rand() \cdot (pbest_{i,d} - x_{i,d}) + c_2 \cdot rand() \cdot (pbest_{g,d} - x_{i,d})$$

$$x_{i,d} = x_{i,d} + v_{i,d}$$

  The process is looped until one of the following conditions are met, a sufficiently good fitness is reached or a maximum number of iterations (generations) are reached.
end

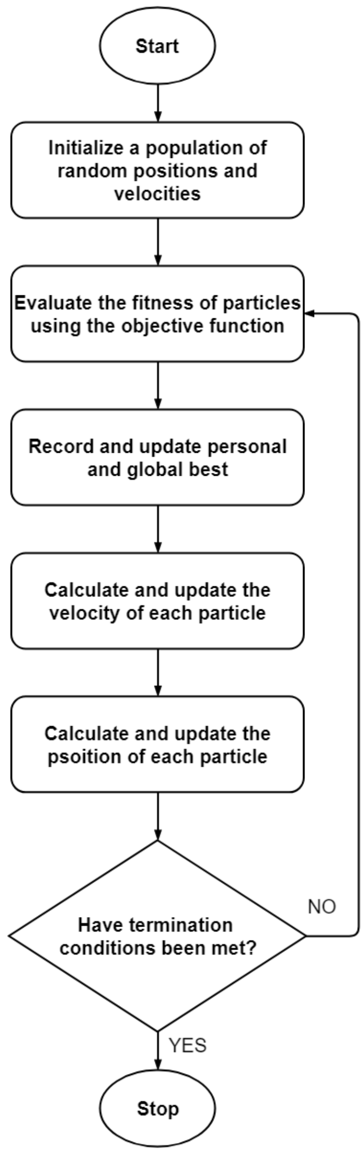

Figure 3.1: Flowchart of the PSO Algorithm

every iteration, the *pbest* value is compared to the corresponding particle's fitness value and updated. This acts as a memory of where the particle was last at its best. Another pair of value and location are recoded which are the overall best particle's fitness value and location. The overall best particle is the particle in the current population that has currently the best fitness value. (Best is usually determined as the lowest or highest fitness value depending on the implementation) These values are known as *gbest* and $pbest_{gd}$. This pair serves as a memory of where the most optimum location is currently at. These recorded values will serve to guide each member to the most optimum location on the search space as seen on the equations at step 14.

The particle's velocity and location are changes in step 14. As we can see, there are many variables involved in the equations. They will be discussed in the next sections.

3.2.1 Background

We first discuss the concepts where the algorithm was based upon. Early computer animations used to simulate a flock of birds by individually giving each bird a script to follow, this includes motion, direction, and speed. Each bird was much like an actor in a play, performing actions under a set of instruction. The problem was that it was not scalable. Animators could not possibly give individual scripts to thousands of birds within a short period. This type of approach is too inefficient. This is why, scientists such as Reynolds[30], Heppner and Grenander [18] have tried to simulate movements of birds and fish using the computational power of computers. They tried to simulate where birds would fly to in every time step or frame in the animation. These simulations were using mathematical and physical concepts to mimic the unpredictable movements of birds when they fly in groups. The initial tests were made such that a population of birds were created, each having its own velocity and initial position on a defined space of definite dimension. These birds were "flying" through the virtual space created by simply adding each bird's velocity to its current position at each time step. Their velocities would change each time step according to the velocities of the nearest neighboring birds to avoid collisions. The initial tests showed that direction and speed were not enough to capture the natural flocking of birds this is because after several time steps, the whole flock would unanimously and uniformly fly through the defined space in an unchanging direction. This resulted in the introduction of a 'craziness' factor in the form of stochastic variables multiplied to the velocities of each bird. This change resulted to simulations looking much more "lifelike".

Let us take, for example, two birds A and B on a real number Cartesian plane. If bird B is flying at a rate of 9 units per second forward and 5 units per second upward and bird B is bird A's neighbor, bird A will change its velocity to match bird B's velocity. Hence, bird A will have a flying rate of 9 units per second forward and 5 units per second upward with each value multiplied to a random number uniformly distributed from 0 to 1. This means, bird A might not fully replicate the velocity that bird B has. This is seen in nature as bird A trying to "approximate" the velocity of bird B in such a way that they will not collide.

The next step towards development was the introduction of a focal point to which the flock would move toward. This was introduced as a "roost" by Heppner[18], typically it is a point in space that indicated where the flock would finally land. Upon simulating this, the birds already have a "lifelike" appearance which therefore allowed the elimination of the 'craziness' factor. It was then noted that birds usually land where there is food, hence the roost was replaced by a vector called the "cornfield vector" which is a two-dimensional array of XY coordinates on the Cartesian plane. Given a known position of food, the birds now changed their velocity according to the distance between their current position and the cornfield vector. Each bird now "remembers" the closest position values it was at during that time step. It also took in consideration the closest position values that any bird in the population has been in. Each bird now changed their velocities with the values that they remember.

The algorithm was then extended to spaces with multiple dimensions. The algorithm was tested from the singular dimesion space R, then to the coordinate system R^2 and finally to the 3-dimensional space, R^3. It was generalized that the algorithm would work in any number of dimensions R^N.

The velocity equation underwent some changes until it became:

$$V[i][d] = c_1 \cdot rand() \cdot (pbest[i][d] - present[i][d]) + c_2 \cdot rand() \cdot (pbest[gbest][d] - present[i][d]) \tag{3.1}$$

where $v[i][d]$ is the d^{th} velocity component of particle i in D dimensions, $rand()$ are the randomly generated stochastic variables, $pbest[i][d]$ is the d^{th} component of the particle's best position in D dimensions, $pbest[gbest][d]$ is the d^{th} component of the population's best particle's position ($gbest$) in D dimensions, $present[i][d]$ is the d^{th} component of the particle's current position in D dimensions, c_1 and c_2 are constant numbers, $x \in (1, 2, 3, \ldots, n)$

Eberhart and Kennedy [10] adopted the term 'swarm' from Millonas under the circumstance that the behavior of the members of the population satisfies the 5 principles of swarm intelligence as proposed by Millonas. These 5 principles are:

1. proximity principle - members are able to carry out simple space and time calculations

2. quality principle - members respond to the quality factors of the environment

3. principle of diverse response - members do not commit it activities along excessively narrow channels

4. principle of stability - members do no change the mode of behavior everytime the environment changes

5. principle of adaptability - members are able to change their mode of behavior when it is worth the computation price

The members of the population satisfy these principles because

1. The population carries out n-dimensional space calculations over a series of time steps

2. Each member responds to the quality of the personal best and global best variables

3. The allocation of responses between personal best and global best ensures diversity of response.

4. The population changes its overall mode of behavior only when the global best changes.

5. The population is adaptive because it does change when the global best changes.

3.2.2 Further Developments

Eberhart and Shi[11] explains that the terms of the velocity vector seen on step 14 of the original algorithm are all important. The first term (v_{id}) being the previous velocity value gives 'memory' to the particle. It keeps the particle at a good position until a better position is found. Without it, the particle will fly towards the centroid of the locations $pbest_{id}$ and $pbest_{gd}$. In addition, without it, the search space will shrink and never grow since it will only move toward the centroid of it's recorded locations $pbest_{id}$ and $pbest_{gd}$. The two terms $c_1 \cdot rand() \cdot (pbest_{id} - x_{id})$ and $c_2 \cdot rand() \cdot (pbest_{gd} - x_{id})$ concerning the personal best and global best comparisons with the current position is necessary to keep the particles from flying in the same direction for every iteration and leaving the search space.

Eberhart and Shi[11] further improved the original algorithm proposed by Eberhart and Kennedy[10] by introducing inertia weight. Inertia weight is responsible for balancing global and local exploration. The new velocity equation becomes

$$v_{id} = v_{id} \cdot w + c_1 \cdot rand() \cdot (pbest_{id} - x_{id}) + c_2 \cdot rand() \cdot (pbest_{gd} - x_{id}) \tag{3.2}$$

where the new variable w is the inertia weight. Eberhart and Shi[11] states that having a high inertia weight ($w > 1.2$) results in more global exploration but less chances of finding the optima because the particles keep exploring new regions in the space. In contrast, having a low inertia weight ($w < 0.8$) will converge to local optima quickly but will not ensure that the global optimum value will be found. Low inertia weight allows for a fine exploration of a region in the space. Having an inertia weight between 0.8 and 1.2 gives the best chances of finding a global optimum but will take a moderate number of iterations. They surmised that it is best to have a high inertia weight in the beginning for extensive global exploration and then reducing the inertia weight gradually through time for a more refined search on local areas. Although the study does give a good background as to the selection of such numbers, in implementing PSO, one must also take in consideration that not all problems are the same hence, implementor must tweak the PSO variables to suit the problems they are trying to solve.

Fixing Convergence

Although PSO is simple in implementation and design, it had certain flaws. It has high computational costs which is given by it slow convergence.[20] Convergence is a problem for PSO because of the restrictions imposed on the velocities of the particle, in addition, although it converges to a point, the particle are ever moving which causes the particles to be in perpetual oscillation around the optima. The population may still converge but due to the perpetual motion, convergence can become a problem if high precision is taken into consideration. The population may not at all converge. Hence, many studies try to solve such problems.

An innovation to the PSO is the introduction of a constriction factor K necessary for ensured convergence introduced by Clerc[4]. The formula then becomes

$$v_{id} = K[v_{id} + c_1 \cdot rand() \cdot (pbest_{id} - x_{id}) + c_2 \cdot rand() \cdot (pbest_{gd} - x_{id})]$$

where $K = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4}|}$, $\varphi = c_1 + c_2$ and $\varphi > 4$.

Chaos Search

Chaos is the characteristic of a non-liner system that includes infinite unstable periodic motions and depends on initial conditions.[29] Due to its uncertainty and stochastic properties, chaotic sequences have been used to replace random generated numbers and to enhance the performance of heuristic optimization algorithms such as GA, PSO and others. There are several chaotic maps available with different properties and characteristics.

The piecewise linear chaotic map (PWLCM) is a simple and efficient chaotic map with good dynamic behavior. The simplest PWLCM is defined by Xiang, Liao and Wong [37],

$$x(t+1) = \begin{cases} x(t)/p, & x(t) \in (0,p) \\ \frac{1-x(t)}{(1-p)}, & x(t) \in [p,1) \end{cases}$$

The PWLCM behaves chaotically in (0,1) when $p \in (0.05) \bigcup (0.5, 1)$. The chaotic variable, x, can be randomly initialized (i.e. $x(0) \ \bigcup (0,1)$) as suggested by Xiang et.al [37] who implemented the PWLCM in PSO to perform chaotic search. They implemented the CPSO (Chaotic PSO) by adding the term $r(2cx - 1)$ to the global best $\hat{y}$. cx is the chaotic variable given by PWLCM and r is a random number taken from the uniform distribution of $(0,1)$. If the resulting vector's objective function value is better, then the global best is replaced, if not, then retain the global best. The velocity function they used for this method is quite different as they have taken inspiration from Clerc and Kennedy [5]:

$$v_{ij} = \chi(v_{ij} + c_1 \cdot r_1 \cdot (y_{ij} - x_{ij}) + c_2 \cdot r_2 \cdot (y_j - x_{ij}))$$

Although it is not very different from the equation of Kennedy et.al.[10], there is no inertial weight present but the variable χ (a.k.a Constricting Factor) is new. χ is added so that the velocity of a particle is throttled such that it does not fly too fast (not having too high of a magnitude for a single time/generation step). χ replaces the need of having to manually set bounds on the magnitude of the velocity. To recall, the velocity of a particle in PSO is usually set to have a bounded magnitude $[vmin\, vmax]$ so that it does not travel too fast through the search space, thereby adding realism and further enhance the ability of each particle to explore the search space thoroughly.

Quantum Mechanics

In Newtonian mechanics a particle has a position and a velocity that determines its trajectory. However, in quantum mechanics, the particle's position and velocity cannot be determined simultaneously according to the uncertainty principle. Hence, the term trajectory is meaningless[34]. Sun et.al.[34] proposed a quantum model of PSO, called QPSO, where particles move according to the following equation,

$$x(t+1) = g \pm \frac{L}{2} ln(\frac{1}{\mu})$$

where g is a local attractor, L is a parameter that must go to zero as $t- >$ inf to guarantee convergence and $\mu \bigcup (0,1)$. L is a very important parameter of QPSO and different methods have been proposed to determine it.[34, 35]

Uncertainty principle states that "the position and the velocity of an object cannot both be measured exactly, at the same time, even in theory. The very concepts of exact position and exact velocity together, in fact, have no meaning in nature."[12]

The uncertainty principle implies that there is no exact trajectory since there is no absolute measurement to the position and/or velocity of an object. This is because there will always be an unknown amount in the measurement given by the precision of the instruments used.

PSO has been improved by combining it with other optimization algorithms as well. These hybrids will be explored in the later sections.

3.3 GA

Genetic Algorithm (GA) is an evolutionary algorithm developed by John Holland et. al.[15] It is based on the mechanics of natural selection and natural genetics, that is, it imitates the processes involved in selection, recombination and evolution. It involves randomness due to the fact that it mimics natural processes, but users can control the degree of randomness that GA exhibits.

The goals of optimization is to improve performance or efficiency towards some goal. However, there is a distinction between the process of improvement and the destination or optimum itself. In this case, GA is the process and is independent of the objective being approached. GA is not focused only solving a single problem. It is a flexible tool used under different circumstances. This robustness makes GA popular among optimization algorithms.

GA was developed by John Holland[15] with the help of his colleagues and students.

Their goal was to (1) abstract and rigorously explain the adaptive process of natural systems and (2) design artificial system software that retains the important mechanisms of natural systems. This approach led to important discoveries in both natural and artificial systems.

3.3.1 Components of GA

The basic algorithm for GA is shown below. A flowchart of the algorithm is also shown on figure 3.2. We now discuss what happens at each part of the algorithm.

Initialization of Population

As we can see, the first step is to generate a population sufficient enough to cover our search space and is limited by the resources at hand. Each member of this population is encoded as an array of values. The number of elements in the array will be determined by the problem and the one who creates the GA. The population size N determines how many chromosomes are in one generation. If there are too few chromosomes, GA will not be able obtain diversity during crossover hence, only a small part of the search space is explored depending on the values of the initial population. On the other hand, if there are too many chromosomes, GA slows down and many of the elements of the initial population tend to be repeated, hence overestimation occurs. After several years of research, it was determined that after some limit (which depends mainly on the encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.[27] This is because the population size becomes too big for the solution space, or the number of computations needed becomes too large and redundant.

Fitness Evaluation

Next is to evaluate the fitness function $f(x)$ for each chromosome x_i in the generation. A fitness function is the function that the algorithm is trying to optimize. The word "fitness" is taken from the evolutionary theory. It tests and quantifies how 'fit' each potential solution is with respect to the problem.[3] It is important to note that the

Algorithm 2: GA Algorithm

Input : Parameters:
Population Size N, Maximum Iterations M, Problem Dimension D, Mutation Probability ρ_m, Crossover Probability ρ_c Boundary Conditions $[l,\, u]$ of each gene

Output: Optimal Solution x_{id}

for $i = 1 : N$ **do**
  Initialize the chromosome i with a uniformly distributed random vector of D dimensions: $x_{i,d} \sim \bigcup(l,\, u)$, $d \in 1, 2, \ldots, D$
end
$j \leftarrow 1$
while $j <= M$ **do**
  Evaluate the fitness function values $F(x_{i,d})$ of each chromosome $x_{i,d}$, $i = 1, 2, \ldots, N$
  `// Create a new population by repeating the following steps`
  for $i = 1 : N$ **do**
    (Selection) Select two parent chromosomes $x_{p_1,d}$ and $x_{p_2,d}$ from the population according to their fitness (the better fitness, the bigger chances of selection)
    `//` p_1 `and` p_2 `are the indices of the selected chromosomes`
    (Crossover)
    if $r \sim \bigcup(0,\, 1) < \rho_c$ **then**
      $o_{1,d} \leftarrow x_{p_1,d}$ CROSS $x_{p_2,d}$
      $o_{2,d} \leftarrow x_{p_2,d}$ CROSS $x_{p_1,d}$
    else
      $o_{1,d} \leftarrow x_{p_1,d}$
      $o_{2,d} \leftarrow x_{p_2,d}$
    end
    (Mutation) if $r \sim \bigcup(0,\, 1) < \rho_m$ **then**
      mutate new offspring at some genes $o_{j,d}$, $j = 1, 2, d \in (1, 2, 3, \ldots, D)$
    end
    (Accepting) Place new offspring o in the new population
  end
  The process is looped until one of the following conditions are met, a sufficiently good fitness is reached or a maximum number of iterations (generations) are reached.
end

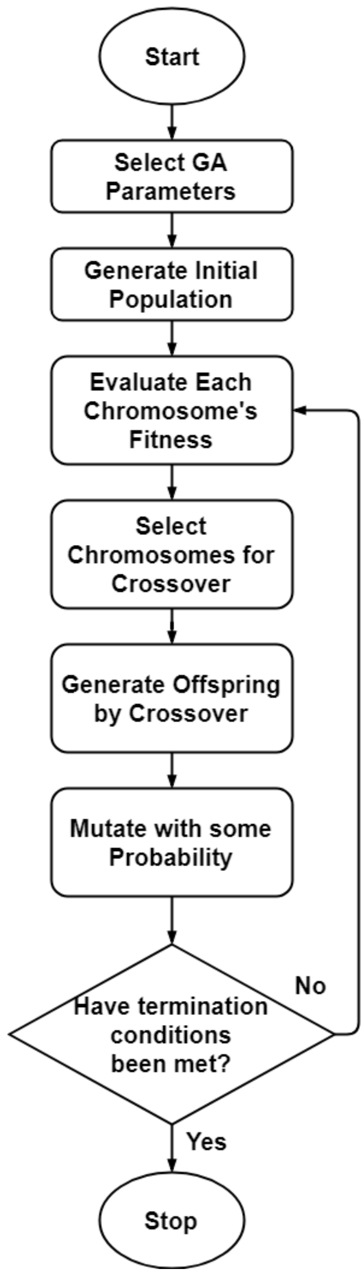

Figure 3.2: Flowchart of the GA Algorithm

fitness function is a large factor in problem solving using GA. The fitness function must be able to define the numerical complexity and constraints that are present in the problem. Choosing the right fitness function will determine computability and complexity usage of the algorithm.

Selection

Selection allows for persistence and propagation of better genes in the next generation based on the current gene pool. The selection process is 'repeating' if it allows re-selection of already selected members. Selection is 'non-repeating' if it does not allow re-selection of members for cross-over. Non-repeating allows retention of other possibly 'good' genes (genes that might lead to better solutions later on) and a slower convergence rate. Repeating selection can lead to a population of individuals that have the same already good genes but differ in only some features. This allows for local exploration, searching for a good solution in a specific area in the search space. In consequence, since the same parents can be selected numerous times, it can lead to generating a population with a uniform genetic make-up.

Examples for selection process are roulette selection and elimination selection. Roulette selection is done by creating a roulette wheel where individuals that have better genes are provided with a lager portion of the whole wheel. The wheel is spun and the corresponding member mapped to the portion of the wheel where the pointer ends at is selected. The process is repeated until there is a good enough number for generating the next population. An example of the roulette wheel is seen on figure 3.3. Elimination selection is done by selecting a number of individuals and pitting them against each other based on their function values. Individuals that have better function values are selected and the process is repeated until there is a good enough number for generating the next population. An example of the elimination selection is seen on figure 3.4.

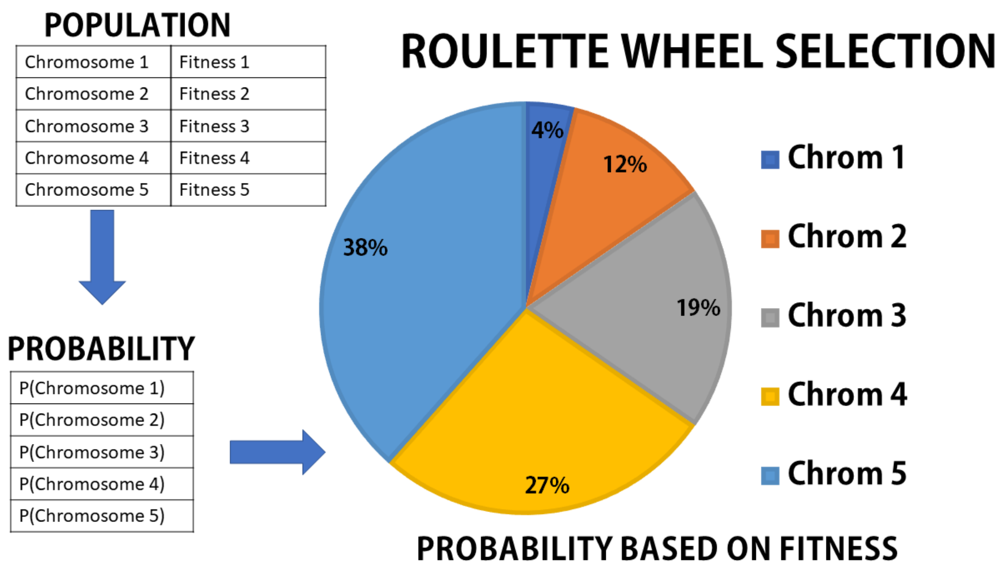

Figure 3.3: Roulette Wheel Selection

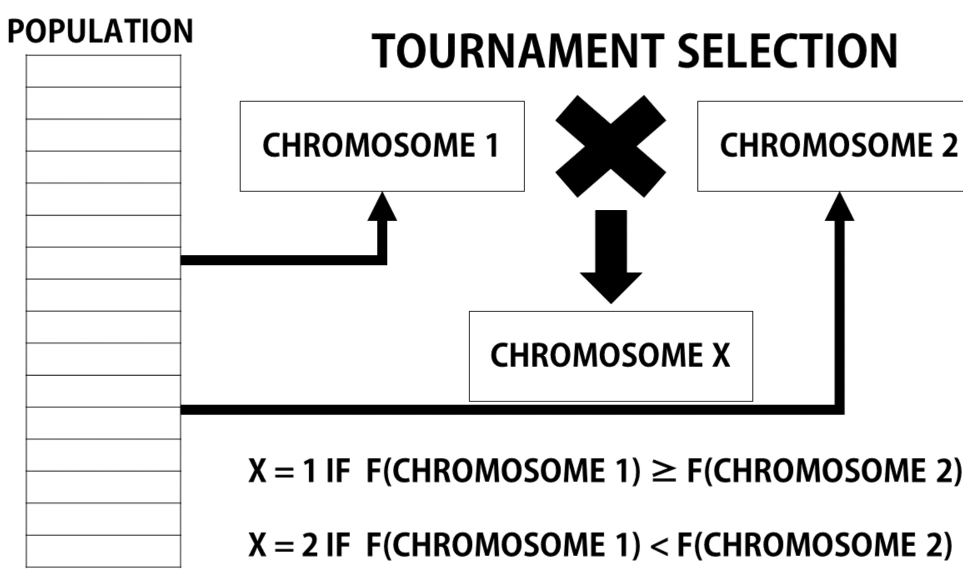

Figure 3.4: Simple Elimination Selection

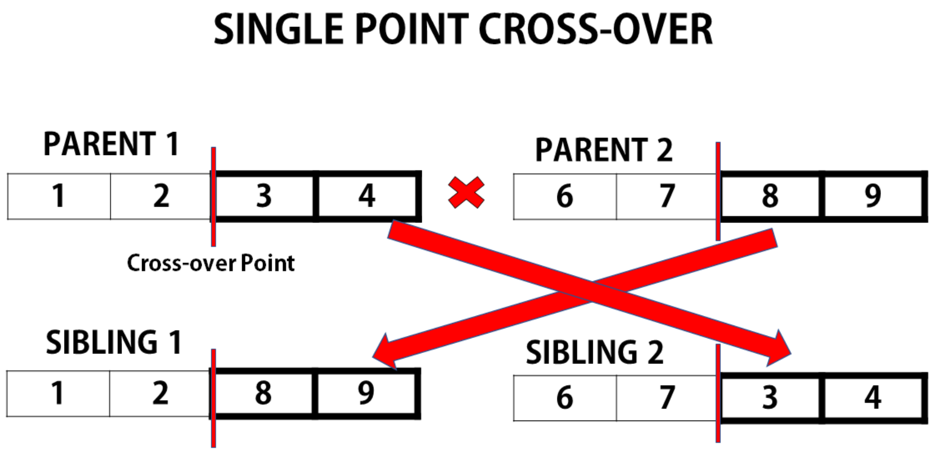

Figure 3.5: Singe Point Cross-Over

Recombination or Cross-over

Cross-over is the process of taking two selected individuals and swapping portions of their genetic make-up to create offspring that have genes from both parents (heredity). The number of points where crossing occurs is determined by the implementor. The cross-over chance is the probability that tells whether recombination occurs for a pair of chromosomes. Cross-over chance is determined by the implementor after some tests. Cross-over mechanism is important because it allows the creation of possibly new solutions from the previous gene pool. This allows exploration over a specific area in the search space. The individuals generated by this process are the members of the next generation. It is up to the implementer how much of the newly generated individuals are chosen. A possible implementation is where offspring that have the better function values may be retained. It is also possible to retain chromosomes form the previous generation. Suppose that we have chromosome A having the genetic make-up $< 1, 2, 3, 4 >$ and chromosome B having the genetic make-up $< 6, 7, 8, 9 >$. If we implement a single point cross-over after the second value we get the offspring $< 1, 2, 8, 9 >$ and $< 6, 7, 3, 4 >$. A visual representation is seen in figure 3.5.

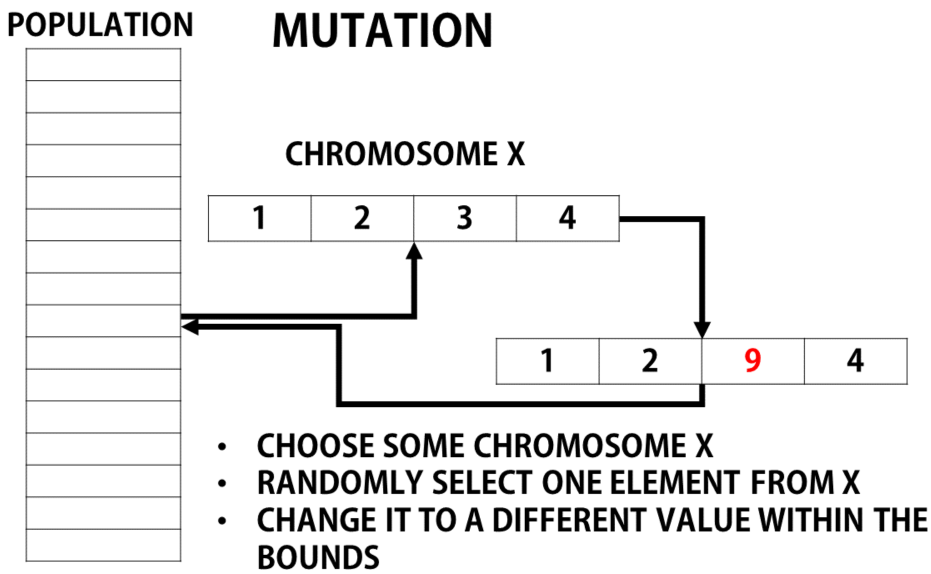

Figure 3.6: Mutation in GA

Mutation

Mutation mechanism is the process wherein some genes of members in the population are replaced by a completely new value. This allows for exploration on possibly 'unexplored' areas in the search space. It helps get the population unstuck from a local optima (optimum solution for a certain area in the search space but may not the most optimal of solutions in the entire search space). Mutation chance is determined by the implementor after some testing. In nature, however, mutation is a rare occasion hence the mutation chance must be low (usually 0.02). A visual representation is seen in figure 3.6. If the chromosomes are bound to have each gene $x_i \in [1, 9]$ where i is from $[1, 4]$. We can see that 3 is replaced by 9 and that both 3 and 9 are still in $[1, 9]$. Note that the number of genes (elements) to be mutated is not limited to one. You can change a few more genes but the number must be small. Mutation is stated to be some minor change in the genes this means that it is up to the implementor to determine the number genes to be manipulated such that it only brings a minor change. When we take binary numbers in consideration, flipping a few bits still creates a minor change if let's say that the chromosome is made up of 30 or 50 genes, then flipping 2-3 bits will not cause a major change provided that they are less significant bits.

Acceptance

Accepting is just evaluating whether or not the generated member can be added to the new population. This can be done through comparing with the parents' fitness values. If the offspring have better fitness values then accept, otherwise reject. This step us usually done after cross-over to check if the siblings generated will replace members in the population based on their fitness.

Replacement

Replace the old population with the new population. We can also choose to retain some of the old population, some of those that have 'good' genes can be kept in the new population. This is called being 'elitist' since it keeps only the fit members of society to move forward.

Termination Conditions

Testing is done through keeping track of the best fitness of each generation. If the fitness is the same for n amount of times and is below a certain acceptable threshold, then we terminate the process. This is considered as a success only if most of the members in the population have the same acceptable fitness, otherwise it is a failure. n is determined by the user. If the population becomes uniform, terminate the process and print out the value. If the fitness is acceptable under a threshold, then it is a success and we say that the population has converged to that point. If the population has become uniform but does not have an acceptable fitness, then it is a convergence but a failure. If a certain number of iterations has been reached and it has not yet converged and has been the same for n times, the process terminates and it is a failure.

3.4 Constrained Optimization Problems

A constrained optimization problem is a problem that is bounded by some limiting factors. Typically, real-worlds problems are always subjected to constraints. For example, if you were to create 3d models of a cube with a single piece of 10x15 inches cardboard. Given that you have limited resources, what is the best way to cut the cardboard in order to have 2 models with the least amount of unused cardboard? That was just a simply problem now, suppose we have a lot of constraints, the problem will becomes more challenging as more constraints are added. Not only the quantity but also the type of the constraints affect the problem as well.

Most real-world optimization problems have constraints of different types which modify the shape of the search space. During the past years, optimization algorithms have been employed to solve such problems. Constraints can be in the form of both equalities and inequalities, they can be discrete and continuous, linear or non-linear, they can also be related to other constraints. Due to these properties, constrained optimization problems are more difficult to solve compared to unconstrained ones. According to Garg[14], Constrained optimization problems are defined as:
Minimize the fitness function $f(x)$ that is subjected to p equality constraints,

$$h_k(x) = 0; k = 1, 2, ..., p$$

and q inequality constraints,

$$g_j(x) \leq 0; j = 1, 2, ..., q$$

where each set of decision variables x is in D dimensions such that $x = [x_1, x_2, x_3, ..., x_D]^T$. Each element of x are bounded as

$$l_i \leq x_i \leq u_i; i = 1, 2, ..., D$$

where l_i and u_i are the minimum and maximum permissible values of each element x_i. In addition, Deb[8] considered that the equality constraints may be converted into inequality constraints since most real world objects are not perfectly accurate in measurement as implied by the uncertainty principle.
Therefore, equality constraints must be formulated such that the function values of the p equality constraints

$$h_k(x) = 0; k = 1, 2, ..., p$$

must be bounded by some allowable precision δ, that is,

$$|h_k(x)| - \delta \leq 0; k = 1, 2, ..., p$$

Notice now that if we set δ to some precision, say 1×10^{-6}, then it can be said that it is approximately equal to 0. If we increase the precision, then it will be nearer to 0 itself, hence, there would not be a very big difference and therefore it may as well be equal to 0. Since the equality constraints have been converted, we now have the new definition, Minimize the fitness function $f(x)$ that is subjected to $M = p + q$ inequality constraints,

$$g_j(x) \leq 0; j = 1, 2, ..., M$$

where each set of decision variables x is in D dimensions such that $x = [x_1, x_2, x_3, ..., x_D]^T$. Each element of x are bounded as

$$l_i \leq x_i \leq u_i; i = 1, 2, ..., D$$

where l_i and u_i are the minimum and maximum permissible values of each element x_i. With that explained, we go on to discuss what feasible and infeasible solutions are. Feasible solutions are solutions that do no violate any constraint while infeasible solutions do. There are many approaches in considering feasible and infeasible solutions when implementing optimization algorithms. Some of these methods include rejection of infeasible individuals, maintaining a feasible population, repairing of infeasible individuals, separation of individuals and constraints, replacement of individuals by their repaired versions and use of decoders.[25]

3.4.1 Penalty Function Approach

In order to solve constrained optimization problems, one may use penalty functions. In using penalty functions, the number of constraint violations are used to punish infeasible solutions so that feasible solutions are much more favored. Unfortunately, penalty functions require parameter tuning for different problems because these parameters are problem-specific.

He and Wang[16] utilized penalty functions for their Co-evolutionary PSO implementation. They used two groups of swarm(s). The first group is used to explore the search space while the other group is used to tweak the penalty function parameters. Each swarm in the exploration group is paired with an individual in the parameter group. The individuals in the parameter group determine the penalty functions to be used by the corresponding swarms in the exploration groups. Hence, the solutions obtained in the exploration group depend upon the parameter group while the parameter group depend on the exploration group for evaluation and tweaking. The process aimed to explore and exploit different search spaces in finding the solution.

On the other hand, Deb[8] proposes a parameter free function in creating a better population to solving constrained optimization problems using GA. These penalty functions do not require values to be set by the user instead, it utilizes the values of the constraint violation themselves. Parameter free penalty function is driven by the new fitness value system,

$$F(x_i) = \begin{cases} f(x_i) & \text{if } x_i \in S \\ f_w + \sum_{j=1}^{M} g_j & \text{if } x_i \notin S \end{cases}$$

where $F(x_i)$ is the penalized objective function value for each individual or particle x_i $i \in 1, 2, 3, ..., N$ in the population, each x_i must be in the S solution space of D dimensions. $f(x)$ is the non-penalized objective function value of x_i, f_w is the worst objective function value among all x_i individuals and each g_j, $j \in 1, 2, 3, ..., M$ is the cost or value of each violated constraint. *The solution space of the problem contains all the viable solutions to the problem which also satisfies each and every constraint present.* If the individual from the population satisfies all conditions, it's fitness value is unchanged but if it does not satisfy the conditions, it's fitness value is changed to that of the worst value added with the values of the violated inequality constraints g_j. This allows the selection operator to give a better chance to feasible solutions by setting the fitness values of infeasible solutions far away from the objective. In PSO and PSO-GA, infeasible solutions means that the population ignores them and only flock towards feasible solutions. In GA, infeasible solutions are ignored or have the least chance of being selected for the selection process.

3.4.2 PSO Fly-back Approach

One method for keeping feasible solutions is to make the particles return to their previous positions.[17] When the individual is to venture upon the infeasible solutions, it "moves back" to its previous position instead of flying to the infeasible solution space. This is done be simple retaining the position it currently is at. One the other hand, while the position remains unchanged, the change in velocity is retained hence, in the next iteration, the particle's velocity becomes shorter and more attuned to facing towards the global best position. This method retains a feasible population but the initial population must be feasible.

3.5 PSO-GA Approach

PSO-GA is a hybrid of PSO and GA. Harish Garg has proposed a PSO-GA[14] which supplements the particular disadvantages of both PSO and GA with the advantages of each. The algorithm attempts to balance the exploration and exploitation ability of both algorithms. Exploration happens in PSO when particle fly through the search space. It is less applicable to GA since the algorithm only utilizes what is current known in the population. It only occurs for GA through Cross-over and Mutation. Exploitation happens in PSO when a particle flies to or near an area containing a possible solution, every other particle in the population will tend to flock towards that area in order to find the solution. PSO's problem is that local optima may trap the whole population. Exploitation happens in GA during the Selection operator, wherein the members with the fittest values have a higher chance of being chosen for Cross-over and Mutation. Hence, more chances of exploring that particular gene pool.

In GA, if an individual is not selected, the information contained by that individual is lost but in PSO, the memory of the previous best position is always available to each individual. Without a selection operator, PSO may waste resources on poorly located individuals. PSO-GA by Garg[14] combines the ability of social thinking in PSO with the local search capability of GA.

PSO's velocity vector guides the population to a certain solution point while GA's selection and cross-over replaces infeasible solutions with feasible ones by creating an individual from the set of feasible solutions.

3.5.1 Parts of PSO-GA

The algorithm for PSO-GA is shown below

1. Set PSO and GA parameters
   - Set current PSO iteration, $PSO_{CurrIt} = 0$ and max iteration PSO_{MaxIt}
   - Set PSO population size PSO_{PopNum}, cognitive and social bias constants c_1 and c_2, maximum and minimum inertial weights w_{max} and w_{min}
   - Set GA parameters, crossover probability GA_{cross}, mutation probability GA_{mut}
   - Set GA parameters: rate of the number of PSO particles affected by GA γ and rate of increasing GA maximum iterations β, maximum and minimum number of individuals to be selected GA_{NumMax} and GA_{NumMin}, maximum and minimum GA population sizes $GA_{MaxPopSize}$ and $GA_{MinPopSize}$, maximum and minimum GA iteration numbers GA_{MinItr} and GA_{MaxItr}
   - Set the PSO dependent GA parameters, number of individuals affected by GA GA_{Num}, GA population size $GA_{PopSize}$ and GA maximum iteration GA_{MaxItr} using the equations

$$GA_{Num} = GA_{NumMax} - (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{NumMax} - GA_{NumMin}) \quad (3.3)$$

$$GA_{PopSize} = GA_{MinPopSize} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{MaxPopSize} - GA_{MinPopSize}) \quad (3.4)$$

$$GA_{MaxItr} = GA_{MinItr} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\beta} \times (GA_{MaxItr} - GA_{MinItr}) \quad (3.5)$$

PSO Section

2. Generate a random population of particles of PSO_{PopNum} members in D dimensions, each with a corresponding random velocity v

3. Increment PSO_{CurrIt} by 1

4. Evaluate each particle's objective function value $F(PSOx)$

5. Update *gbest* and *pbest* positions and values of each $PSOx_i$ in the population $(i \in 1, 2, 3, \ldots, PSO_{PopNum})$

6. Update each particle's velocity and position with the equations,

$$w = w_{max} - (w_{max} - w_{min}) \times (\frac{PSO_{CurrIt}}{PSO_{MaxIt}}) \tag{3.6}$$

$$v_i = v_i \times w + c_1 \times rand() \times (pbest_i - PSOx_i) + c_2 \times rand() \times (pbest_g - PSOx_i) \tag{3.7}$$

   where $i \in 1, 2, 3, \ldots, PSO_{PopNum}$ and g is position/individual in the PSO population that is currently designated as global best ($gbest$) individual

$$PSOx_i = PSOx_i + v_i \tag{3.8}$$

GA Section

7. Set the number of currently selected individuals $GA_{CurrNum} = 0$
8. Increment $GA_{CurrNum}$ by 1
9. Choose a random position/individual $PSOx_s$ from the PSO population.
10. Generate a random population of $GA_{PopSize}$ individuals in the same D dimensions.
11. Set the first individual GAx_1 in the GA population to be a randomly selected individual $PSOx_s$ from the PSO particle population.
12. Set the current GA iteration $GA_{CurrItr} = 0$
13. Increment $GA_(CurrItr)$ by 1
14. Perform elitism
    - set the replacing individual GA_{rep} as the randomly selected PSO particle $PSOx_s$ if $GA_{CurrNum} = 0$
    - otherwise, check each individual in the current GA population, if $F(GAx_i)$ is less fit than $F(PSOx_s)$, then replace GAx_i with $PSOx_s$

$$GAx_i = \begin{cases} PSOx_s & \text{if } F(PSOx_s) < F(GAx_i) \\ GAx_i & \text{otherwise} \end{cases} \quad i \in 1, 2, \ldots, GA_{PopSize}$$

15. Perform selection, crossover and mutation to generate the next GA population

16. Evaluate the penalizing objective fitness values $F(GAx_i)$ for each individual in the GA population

17. Check if maximum GA iterations is reached
    - If reached, proceed to step 18
    - otherwise, go back to step 13

18. Replace the selected PSO particle $PSOx_s$ with the best individual in the GA population

19. Check if the maximum number of replacements have occurred
    - If reached, proceed to step 20
    - otherwise, go back to step 9

20. Update the PSO dependent GA parameters using equations (3.3), (3.4) and (3.5)

21. Check if the maximum number of PSO iterations have been reached or if the population has converged
    - If reached, end
    - otherwise, go back to step 3

The flowchart of the algorithm is shown on figure 3.7

As you can see, the algorithm follows the both PSO and GA algorithms in succession. PSO is first done to the population to obtain points across the search space. GA is then applied to some of the best individuals. This is done to replace the worst individuals in the population with those closer to the better ones.

After forming the new population with PSO, some of the individuals in the population will get replaced. Some not all because if we have a huge population, it would take a long time to complete. This number is given by GA_{Num}. After selecting the best individuals from the population, the algorithm aims to create a new population by replacing points in the current population with better points via the genetic principles, selection, cross-over and mutation. After all selected individuals have been processed, we change the GA variables, $GA_{PopSize}$ and GA_{MaxItr} which are for the population size in GA and

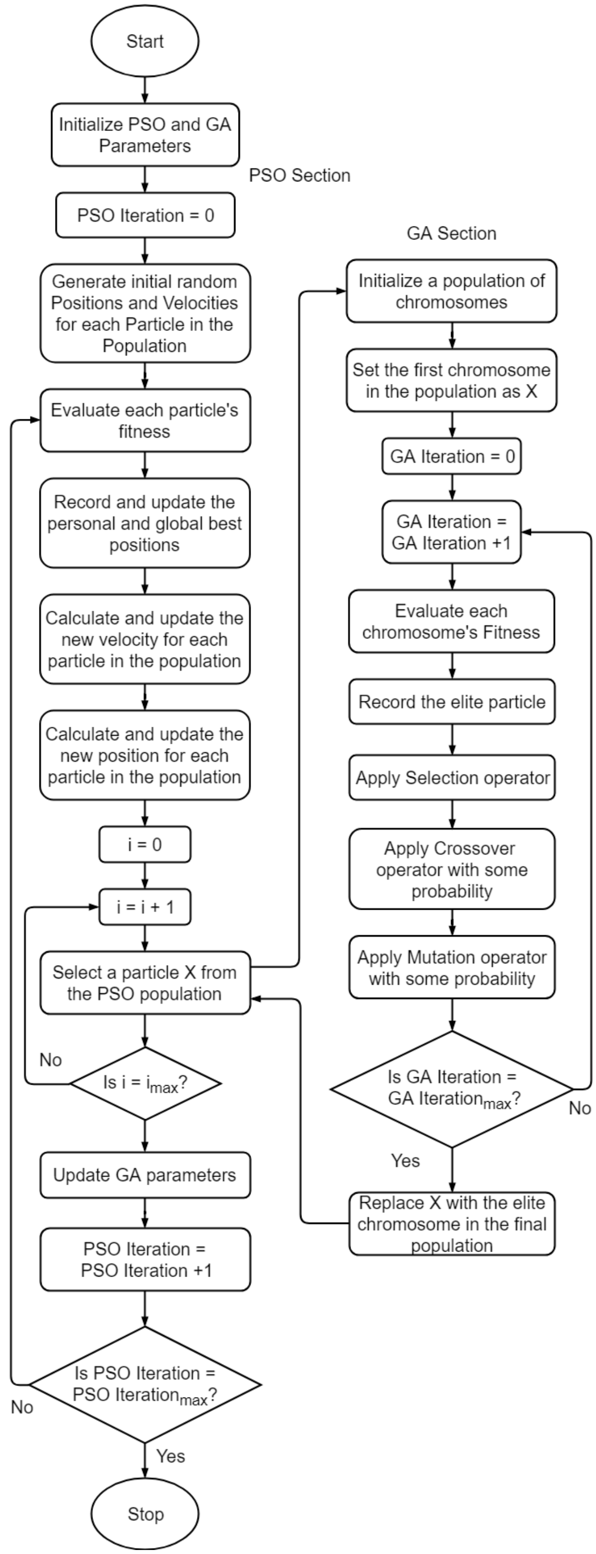

Figure 3.7: Flowchart of PSO GA Algorithm

the maximum iterations done for GA respectively by the equations (3.3), (3.4) and (3.5).

Judging from the equations 3.3, 3.4 and 3.5, GA_{Num} will initially be GA_{NumMax} and slowly become GA_{NumMin} as the number of iterations increases. This is because the fraction PSO_{CurrIt}/PSO_{MaxIt} is raised to γ which is a positive whole number as given by Garg[14], hence, the whole term $(PSO_{CurrIt}/PSO_{MaxIt})^{\gamma}$ will initially be very small and eventually will be equal to 1 when $PSO_{CurrIt} = PSO_{MaxIt}$.

This is also the case for both $GA_{PopSize}$ and GA_{MaxItr}. $GA_{PopSize}$ will initially start equal to $GA_{MinPopSize}$ then slowly become $GA_{MaxPopSize}$. GA_{MaxItr} will initially start equal to GA_{MinItr} then slowly become GA_{MaxItr}. Since the factors will be in fractions, there is a need to get the floor values of GA_{Num}, $GA_{PopSize}$ and GA_{MaxItr}. This is because GA_{Num}, $GA_{PopSize}$ and GA_{MaxItr} must be positive integers because they dictate array sizes. However, in the case of Inertial Weight w, which changes according to the equation $w = w_{max} - (w_{max} - w_{min})(PSO_{CurrIt}/PSO_{MaxIt})$, it is most of the time a fraction. Garg[14] started $w = 0.9$ initially then becoming $w = 0.4$ as the number of iterations increases.

Chapter 4

Methodology

In this chapter, we first discuss the model of the problem then show how the hybrid PSO-GA algorithm of Harish Garg[14] is implemented. Then we test the effectiveness of the algorithm in solving vehicle routing problems using the VRPTW model used by Liu et.al.[22].

4.1 Problem Model

We now consider developing the specific model for the Baguio City waste collection routing problem. We start by reevaluating what we know about the current waste collection system in Baguio City. The data on the vehicles and working hours was provided by the Sold Waste Management Division of Baguio City.

- Each driver works for 9 hours each working day on different shifts; morning, afternoon and night.
- Each driver is assigned a 5-day work schedule on different sets of days.
- Each vehicle is assigned to service about 7 to 8 Barangays each day.
- There are, as of June 2018, currently a total of 19 waste collection vehicles. Two of which act as quick response vehicles, these are operated by two teams responsible for collecting the extra amount of waste that is left when a waste collection vehicle becomes too full to collect all of the garbage on-site.
- There are four kinds of vehicles used for waste collection. Most of the vehicles have an approximate capacity of 12 cubic meters.
- Each vehicle has two main partitions for biodegradable and residual waste, however, the partitioning is not fixed.
- Each vehicle start and ends at the Irisan ERS/MRF.
- Each vehicle is empty before leaving the ERS/MRF.

- The vehicles are full when they return to the ERS/MRF but their load is deposited at the site for final segregation.
- After being sorted, residual waste is brought to the Garbage Transfer Station at Barangay Dontogan where it will be gathered and loaded onto vehicles that transport it to Capas, Tarlac.
- Biodegradable waste remains at Irisan ERS/MRF while the rest of the recyclables are either given away or sold for the compensation of volunteer sorters.
- There are 129 known Barangays (Villages) in the City that are serviced by the General Services Office - Solid Waste Management Division.
- No time windows are alloted to each collection site due to variability of traffic, road availability, weather conditions, and quantity of waste.

4.2 Waste Collection Vehicle Routing Problem Model

The objective in Waste Collection Vehicle Routing Problem is to determine a feasible set of routes that minimizes the total cost involved in waste collection with the following constraints:

1. All vehicles begin at and return to the depot;
2. All vehicles are homogeneous, they have the same maximum capacity;
3. A waste collection site is visited by only one vehicle;
4. The total amount of waste collected by vehicle must not exceed its maximum;
5. Distances between the depot, collection sites and the disposal site are determined;
6. We assume that the disposal site is the same as the depot. This is because the waste collected by trucks will have to be sorted at the ERS-MRF at Irisan before it is transported to the Garbage Transfer Station (GTS). The GTS is not part of the scope of this problem because the job of handling the transfer from Baguio to Tarlac handed to a different group;

7. The demand at each collection site should be less than the maximum capacity of the vehicle. Note that any excess amount at a site will always be covered by the quick response teams.

We represent our network of collection sites and ERS-MRF depot/disposal site as a complete undirected graph $G = (V, E)$ of V vertices and E edges.

The set of vertices V encapsulates the set of waste collection sites (V^c) and the single depot also considered as the single disposal site (V^d), that is $V = \{V^d \cup V^c\}$. The number of vertices is therefore $|V| = |V^d| + |V^c| = 1 + n = N$ where n is the number of waste collection sites.

$$V = \{v_i\}, i \in 0, 1, 2, \ldots, n$$

where

$$v_i = \begin{cases} v_0 & \text{is the Depot} \\ v_1, v_2, \ldots, v_n & \text{are the Collection Sites} \end{cases}$$

Each vertex $v_i \in V$ is associated with a demand q_i equivalent to the amount of garbage to be collected in cubic meters.

$$q_i = \begin{cases} q_0 = 0 \text{ m}^3 \\ q_1, q_2, \ldots, q_n \in \mathbb{R} \\ \text{ specifically } \in [g, Q] \text{ m}^3 \end{cases}$$

where g and Q are the lower and upper bounds of the amount of garbage that can be generated at a collection site i, $i = 1, 2, \ddots, n$ moreover, Q is the maximum carrying capacity of a vehicle, defined later.

The set of edges

$$E = \{(v_i, v_j) | v_i, v_j \in V,\ i, j \in 0, 1, 2 \ldots, n\}$$

The edge $(v_i, v_j) \in E$ connects an arbitrary pair of vertices v_i, v_j in graph G.

Each edge $(v_i, v_j) \in E$ is associated to a distance $d_{i,j}$ in kilometers. Let $K = \{k_i\}, i \in 1, 2, 3, \ldots, m$ be the set of waste collection vehicles. The number of vehicles m varies depending on the route constructed however, we set that $1 \leq m \leq n$. There would always be one vehicle in any route and the maximum number of vehicles that can used in a route is equal to the number of collection sites n, this happens when every collection site is serviced exclusively by its own waste collection vehicle.

Let Q be the maximum carrying capacity of any vehicle $k \in K$. This is the maximum amount of garbage that can be collected and carried by a vehicle along it's path.

The decision variables of the model depend on the vehicle capacity Q and the waste quantity at the next waste collection site it visits. These are modeled as follows:

$$X_{i,j,l} = \begin{cases} 1, & \text{if vehicle } k_l \text{ can travel from vertex } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where $i, j \in 0, 1, 2 \ldots, n.$ and $l \in 1, 2, \ldots, m.$

$$A_{i,j,l} \in \mathbb{R}, \text{ specifically } \in [0, Q] \tag{4.2}$$

where each element in A is the accumulated amount collected by vehicle $k_l \in K$ when moving between v_i and v_j where $l \in 1, 2, \ldots, m$ and $v_i, v_j \in V$, $i, j \in 0, 1, 2 \ldots, n$.

$$Y_{i,l} = \begin{cases} 1, & \text{if vertex } v_i \text{ is visited by vehicle } k_l \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where $i \in 1, 2 \ldots, n$ and $l \in 1, 2, \ldots, m$. Note that we do not consider the depot here because it is bound to be visited more than once by any vehicle.

We can say that X is an $N \times N \times m$ matrix where each $N \times N$ is the adjacency matrix of the route of vehicle k_l, $l \in 1, 2, ..., m$. Hence, it is the adjacency matrix of a subgraph of G where either none, few, many or all of the edges may have been taken out. Moreover, if we combine all of the m $N \times N$ matrices, we come-up with a denser subgraph of G or G itself. It follows that A is also the same $N \times N \times m$ matrix where instead of taking binary variables, it takes on values that represent the accumulative amount of waste collected by a vehicle during it's run through edge (v_i, v_j). Y is an $n \times m$ matrix that acts more like a checklist that shows which waste collection sites $v_i \in V^c$ were visited by vehicle $k_l \in K$.

Our aim is to minimize operational costs. Specifically, we want to minimize the total amount of travel cost while minimizing the fleet size (number of vehicles used). We already know that minimizing travel cost is about selecting the best set of ways that provide us the least amount of expenses between any two points. We now focus on reducing fleet size. We want to know how we can maximize the use of each and every vehicle in the fleet which will be discussed after the model. Our objective function is represented by the equation:

$$\min F(X,A,m) = \alpha_1 \cdot \left(\sum_{l=1}^{m} \sum_{i=0}^{n} \sum_{j=0}^{n} X_{i,j,l} \cdot d_{i,j} \right) + \alpha_2 \cdot \sum_{l=1}^{m} \sum_{i=1}^{n} A_{i,0,l} + \alpha_3 \cdot m \tag{4.4}$$

Where α_1 is the constant which converts distance to cost, α_2 is the constant which converts the total waste collected by all vehicles to cost, and α_3 is the constant which converts the number of vehicles to cost.

In order to make satisfy our assumptions, we subject our objective function following constraints:

$$\sum_{i=1}^{n} \sum_{l=1}^{m} X_{i,j,l} = 1, \qquad \forall j = 1, 2, \ldots, n \tag{4.5}$$

$$\sum_{i=1}^{n} Y_{i,l} = \sum_{i=1}^{n} X_{i,j,l}, \qquad \forall l = 1, 2, \ldots, m; j = 1, 2, \ldots, n \tag{4.6}$$

$$\sum_{j=0}^{n} X_{0,j,l} = 1, \qquad \forall l = 1, 2, \ldots, m \tag{4.7}$$

$$\sum_{i=0}^{n} X_{i,0,l} = 1, \qquad \forall l = 1, 2, \ldots, m \tag{4.8}$$

$$\sum_{j=1}^{n} A_{0,j,l} = 0, \qquad \forall l = 1, 2, \ldots, m \tag{4.9}$$

$$\sum_{l=1}^{m} \sum_{j=1}^{n} A_{i,j,l} \leq Q, \qquad \forall i = 0, 1, \ldots, n \tag{4.10}$$

$$\sum_{l=1}^{m} (A_{j,h,l} - A_{i,j,l}) = \sum_{l=1}^{m} X_{i,j,l} \cdot q_j, \qquad \forall i, h = 0, 1, \ldots, n; j = 1, 2, .., n-1 \tag{4.11}$$

$$dist_{i,j} = dist_{j,i}, \qquad \forall i = 0, 1, \ldots, n; j = 0, 1, \ldots, n \tag{4.12}$$

$$X_{i,j,l} \in 1, 0 \tag{4.13}$$

$$Y_{i,l} \in 1, 0 \tag{4.14}$$

$$A_{i,j,l} \in \mathbb{R} \tag{4.15}$$

Constraint (4.5) specifies that collection site v_i is visited by not more than one vehicle k_l and (4.6) specifies that a collection site v_i is in the route of vehicle k_l. Since the values of each X_{ijl} is 1 if vehicle k_l moves from vertex v_i to v_j and 0 otherwise, then if we get the sum of the values, we will know how many times v_i is visited by all vehicles. However, we assumed that vehicles only visit each collection site once, hence, the sum must be equal to one.

Constraints (4.7) and (4.8) imposes that each vehicle $k_l \in K$ must start and end at the depot.

Constraint (4.9) imposes that each vehicle $k_l \in K$ must have no accumulated waste before leaving and returning to the depot.

Constraint (4.10) imposes that the accumulated amount of any vehicle $k_l \in K$ traveling between any pair of vertices v_i and v_j must be less than the maximum capacity.

Constraint (4.11) imposes that the vehicle k_l completely collects all waste when it visits vertex v_j.

Constraint (4.12) imposes that the total distance traveled from vertex v_i to vertex v_j must be the same when the vehicle k_l travels from vertex v_j to vertex v_i.

Constraints (4.13), (4.14), and (4.15) define the domain of the decision variables.

We now explain the values of the three constants α_1, α_2, and α_3. We set that the amount of waste is in cubic meters and our distances are in kilometers, we calculate the total cost in terms of operational cost in Philippine Pesos (Php). We first discuss the value of α_1. In order to convert the total distance covered in operational cost, we must know how much amount of fuel in liters is needed to travel that amount of distance. Then we convert the liters of fuel into operational cost. Hence, our conversion is done as follows:

$$\text{Total Distance } \cancel{\text{Km}} \times \frac{\tau \cancel{\text{ Liter}}}{\cancel{\text{Km}}} \times \frac{\lambda \text{ Pesos}}{\cancel{\text{Liter}}} = \text{ Total Distance} \cdot \tau \cdot \lambda \text{ Pesos}$$

where τ is the fuel efficiency of the vehicle and λ is the cost of a liter of fuel. Fuel efficiency τ is obtained by calculating the average daily fuel consumption and travel distance of the vehicle. This data was obtained through the Monthly Report of Fuel Consumption and Official Travel produced by the Solid Waste Management Division. This report consists of the distance traveled by the vehicle and the amount of gas used for the day. Measuring distance traveled and fuel consumption is done by the odometer of the vehicle. These measurements are recorded by the driver before and after vehicle use. Fuel efficiency of the vehicle used in this model is approximately 0.27 Liters per Kilometers. The cost of the liter of fuel is obtained by checking the gas prices at the petrol stations for a particular span of time. Specifically, we recorded the diesel prices from June 28 to July 2 of 2018 and observed that the diesel costs 46.20 Philippine Pesos (Php) per liter on all five days. Hence, $\alpha_1 = \tau \cdot \lambda = 0.27 \cdot 46.20 = 12.474$ pesos per kilometer.

As for α_2 and α_3, these constants are for vehicle minimization. α_3 is the salary of a driver who drives vehicle $l \in K$. The conversion from number of vehicles to operational cost in Php is done as follows.

$$m \cancel{\text{ vehicles}} \times \frac{1 \cancel{\text{ driver}}}{\cancel{\text{vehicle}}} \times \frac{\alpha_2 \text{ Pesos}}{\cancel{\text{driver}}} = m \cdot \alpha_2 \text{ Pesos}$$

The value of α_3 is given by the average salary of drivers. The salaries of drivers depend mainly upon their years of service. The range of the salaries of the drivers are from Php 480 to Php 1200 and above. Hence, $\alpha_3 = \frac{480+1200}{2} = 840$. Lastly, we discuss the value of α_2. Recyclable and reusable waste can be sold by waste collectors to companies that need these materials. An example of this are the glass bottles which can be remelted and molded for either the same use or a different one. α_2 is the amount of money obtained when the recyclables of a fully loaded vehicle is sold. Generally, all the recyclable materials sum up to about Php 400 for a fully loaded truck. Therefore, for the second term of our objective function (4.4), we obtain the sum of all the waste collected by all vehicles and divide that amount by the maximum capacity of a vehicle so that we know how much truck loads of waste was collected. We then multiply that to α_2 so that we know how much money was obtained through selling the recyclables. However, in reality, this amount of money does not go to the management. This amount

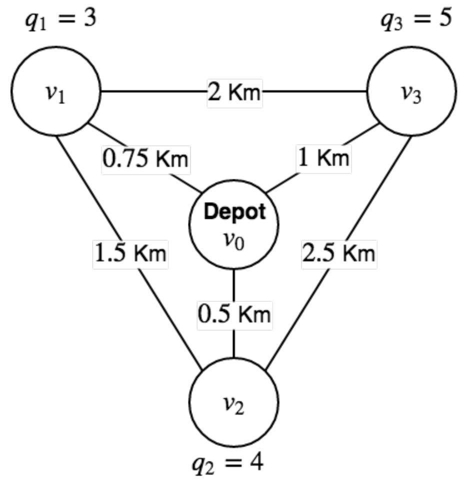

Figure 4.1: Graph of the Basic Example

is given to the volunteers who sort and load the waste on-site. These volunteers are not directly paid by the government but they obtain compensation for their labor through the money obtained from selling recyclable and reusable waste. The conversion from amount of waste to peso is done as follows.

$$\frac{\text{Total Collected Waste } \cancel{m^3}}{Q\frac{\cancel{m^3}}{\cancel{\text{vehicle}}}} \times \frac{\alpha_1 \text{ Pesos}}{\cancel{\text{vehicle}}} = \frac{\alpha_1 \cdot \text{ Total Collected Waste}}{Q} \text{ Pesos}$$

Do note that the second term of the objective function (4.4) is dependent on the amount of waste collected however, when we only talk about feasible solutions, this value will become constant for all feasible solutions because all waste is collected by the fleet. The second term generally depends on the instance of the problem. When the waste to be collected is large, then so is the amount of recyclables recovered.

We have now established the model for the problem however, we show the reasoning behind the added cost of waste collected and driver salaries. If the problem was just about obtaining the shortest distance, then the problem becomes a Traveling Salesman Problem. The problem would be as simple as finding the shortest connections between nodes by using either the Dijsktra's algorithms. Therefore, the amount of vehicles is ignored. We show an example where a full garbage truck is better than using multiple garbage trucks. If we have graph G in figure 4.1. Then we know that the solution to the obtaining the minimum distance is to visit each collection site on different trips, given by the route $0 \to 1 \to 0 \to 2 \to 0 \to 3 \to 0$. That is, if we only consider the first term of our objective function (4.4), then the solution would be the sum of the hadarmard products of each of the vehicle's routes and the distance matrix. Let α_1 be the same value stated above.

We know that each collection site is exclusively serviced by their own vehicles. Therefore we have three waste vehicle collection vehicles traveling from the depot, servicing the node v_1, v_2 and v_3 respectively, then return to the depot after collection. The edges they used are given as follows.

$$X_{i,j,1} = \begin{array}{c c} v_i \diagdown v_j & \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$X_{i,j,2} = \begin{array}{c c} v_i \diagdown v_j & \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} & \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$X_{i,j,3} = \begin{array}{c c} v_i \diagdown v_j & \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} & \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The distance matrix of the graph in figure 4.1 is given below.

$$d_{i,j} = \begin{array}{c c} v_i \diagdown v_j & \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} & \begin{pmatrix} 0 & 0.75 & 0.5 & 1.0 \\ 0.75 & 0 & 1.5 & 3.0 \\ 0.5 & 1.5 & 0 & 2.5 \\ 1.0 & 3.0 & 2.5 & 0 \end{pmatrix} \end{array}$$

Therefore, the cost of traveling this route is given by

$$\begin{aligned} F =& \alpha_1 \cdot \left(\sum_{i=0}^{3} \left(\sum_{j=0}^{3} (X_{i,j,1} \cdot d_{i,j}) \right) + \sum_{i=0}^{3} \left(\sum_{j=0}^{3} (X_{i,j,2} \cdot d_{i,j}) \right) + \sum_{i=0}^{3} \left(\sum_{j=0}^{3} (X_{i,j,3} \cdot d_{i,j}) \right) \right) \\ =& \alpha_1 \cdot \left[\left(\sum_{j=0}^{3} (X_{0,j,1} \cdot d_{0,j}) \right) + \left(\sum_{j=0}^{3} (X_{1,j,1} \cdot d_{1,j}) \right) + \left(\sum_{j=0}^{3} (X_{2,j,1} \cdot d_{2,j}) \right) + \right. \\ & \left(\sum_{j=0}^{3} (X_{3,j,1} \cdot d_{3,j}) \right) + \left(\sum_{j=0}^{3} (X_{0,j,2} \cdot d_{0,j}) \right) + \left(\sum_{j=0}^{3} (X_{1,j,2} \cdot d_{1,j}) \right) + \\ & \left(\sum_{j=0}^{3} (X_{2,j,2} \cdot d_{2,j}) \right) + \left(\sum_{j=0}^{3} (X_{3,j,2} \cdot d_{3,j}) \right) + \left(\sum_{j=0}^{3} (X_{0,j,3} \cdot d_{0,j}) \right) + \\ & \left. \left(\sum_{j=0}^{3} (X_{1,j,3} \cdot d_{1,j}) \right) + \left(\sum_{j=0}^{3} (X_{2,j,3} \cdot d_{2,j}) \right) + \left(\sum_{j=0}^{3} (X_{3,j,3} \cdot d_{3,j}) \right) \right] \end{aligned}$$

$$
\begin{aligned}
=&\alpha_1 \cdot \{(X_{0,0,1} \cdot d_{0,0}) + (X_{0,1,1} \cdot d_{1,1}) + (X_{0,2,1} \cdot d_{1,2}) + (X_{0,3,1} \cdot d_{1,3}) + (X_{1,0,1} \cdot d_{1,0}) + \\
&(X_{1,1,1} \cdot d_{1,1}) + (X_{1,2,1} \cdot d_{1,2}) + (X_{1,3,1} \cdot d_{1,3}) + (X_{2,0,1} \cdot d_{2,0}) + (X_{2,1,1} \cdot d_{2,1}) + \\
&(X_{2,2,1} \cdot d_{2,2}) + (X_{2,3,1} \cdot d_{2,3}) + (X_{3,0,1} \cdot d_{3,0}) + (X_{3,1,1} \cdot d_{3,1}) + (X_{3,2,1} \cdot d_{3,2}) + \\
&(X_{3,3,1} \cdot d_{3,3}) + (X_{0,0,2} \cdot d_{0,0}) + (X_{0,1,2} \cdot d_{1,1}) + (X_{0,2,2} \cdot d_{1,2}) + (X_{0,3,2} \cdot d_{1,3}) + \\
&(X_{1,0,2} \cdot d_{1,0}) + (X_{1,1,2} \cdot d_{1,1}) + (X_{1,2,2} \cdot d_{1,2}) + (X_{1,3,2} \cdot d_{1,3}) + (X_{2,0,2} \cdot d_{2,0}) + \\
&(X_{2,1,2} \cdot d_{2,1}) + (X_{2,2,2} \cdot d_{2,2}) + (X_{2,3,2} \cdot d_{2,3}) + (X_{3,0,2} \cdot d_{3,0}) + (X_{3,1,2} \cdot d_{3,1}) + \\
&(X_{3,2,2} \cdot d_{3,2}) + (X_{3,3,2} \cdot d_{3,3}) + (X_{0,0,3} \cdot d_{0,0}) + (X_{0,1,3} \cdot d_{1,1}) + (X_{0,2,3} \cdot d_{1,2}) + \\
&(X_{0,3,3} \cdot d_{1,3}) + (X_{1,0,3} \cdot d_{1,0}) + (X_{1,1,3} \cdot d_{1,1}) + (X_{1,2,3} \cdot d_{1,2}) + (X_{1,3,3} \cdot d_{1,3}) + \\
&(X_{2,0,3} \cdot d_{2,0}) + (X_{2,1,3} \cdot d_{2,1}) + (X_{2,2,3} \cdot d_{2,2}) + (X_{2,3,3} \cdot d_{2,3}) + (X_{3,0,3} \cdot d_{3,0}) + \\
&(X_{3,1,3} \cdot d_{3,1}) + (X_{3,2,3} \cdot d_{3,2}) + (X_{3,3,3} \cdot d_{3,3})\} \\
=&\alpha_1 \cdot \{(0 \cdot 0) + (1 \cdot 0.75) + (0 \cdot 0.5) + (0 \cdot 1.0) + (1 \cdot 0.75) + (0 \cdot 0) + (0 \cdot 1.5) + (0 \cdot 2.0) + \\
&(0 \cdot 0.5) + (0 \cdot 1.5) + (0 \cdot 0) + (0 \cdot 2.5) + (0 \cdot 1.0) + (0 \cdot 2.0) + (0 \cdot 2.5) + (0 \cdot 0) + \\
&(0 \cdot 0) + (0 \cdot 0.75) + (1 \cdot 0.5) + (0 \cdot 1.0) + (0 \cdot 0.75) + (0 \cdot 0) + (0 \cdot 1.5) + (0 \cdot 2.0) + \\
&(1 \cdot 0.5) + (0 \cdot 1.5) + (0 \cdot 0) + (0 \cdot 2.5) + (0 \cdot 1.0) + (0 \cdot 2.0) + (0 \cdot 2.5) + (0 \cdot 0) + \\
&(0 \cdot 0) + (0 \cdot 0.75) + (0 \cdot 0.5) + (1 \cdot 1.0) + (0 \cdot 0.75) + (0 \cdot 0) + (0 \cdot 1.5) + (0 \cdot 2.0) + \\
&(0 \cdot 0.5) + (0 \cdot 1.5) + (0 \cdot 0) + (0 \cdot 2.5) + (1 \cdot 1.0) + (0 \cdot 2.0) + (0 \cdot 2.5) + (0 \cdot 0)\} \\
=&\alpha_1 \cdot \{0.75 + 0.75 + 0.5 + 0.5 + 1.0 + 1.0\} = 4.5 \cdot \alpha_1 = 4.5 \cdot 12.474 = \text{ Php } 56.133
\end{aligned}
$$

We see that the cost of the shortest route gives Php 56.133. We now add the second and third terms of the our objective function (4.4). Let the maximum capacity of the vehicles for this example be $Q = 12\text{m}^3$, and let $\alpha_2 = 400$ and $\alpha_3 = 840$. The unique feasible solutions of this graph and their function values are as follows:

$$\begin{aligned}
v_0 \to v_1 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 = \quad & 12.474 \cdot 4.50 - 400 \cdot \frac{9}{9} + 840 \cdot 3 = & \text{Php } 2,176.1330 \\
v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_0 = \quad & 12.474 \cdot 4.75 - 400 \cdot \frac{9}{9} + 840 \cdot 2 = & \text{Php } 1,339.2515 \\
v_0 \to v_1 \to v_3 \to v_0 \to v_2 \to v_0 = \quad & 12.474 \cdot 4.75 - 400 \cdot \frac{9}{9} + 840 \cdot 2 = & \text{Php } 1,339.2515 \\
v_0 \to v_2 \to v_3 \to v_0 \to v_1 \to v_0 = \quad & 12.474 \cdot 5.50 - 400 \cdot \frac{9}{9} + 840 \cdot 2 = & \text{Php } 1,348.6070 \\
v_0 \to v_1 \to v_2 \to v_3 \to v_0 = \quad & 12.474 \cdot 5.75 - 400 \cdot \frac{9}{9} + 840 \cdot 1 = & \text{Php } 511.7255 \\
v_0 \to v_3 \to v_1 \to v_2 \to v_0 = \quad & 12.474 \cdot 5.50 - 400 \cdot \frac{9}{9} + 840 \cdot 1 = & \text{Php } 502.3700 \\
v_0 \to v_2 \to v_3 \to v_1 \to v_0 = \quad & 12.474 \cdot 5.75 - 400 \cdot \frac{9}{9} + 840 \cdot 1 = & \text{Php } 511.7255
\end{aligned}$$

The best solution changes because the route that gives us the shortest distance utilizes three vehicle.The best solution in this case is the route where only one vehicle is used, this is given by $0 \to 3 \to 1 \to 2 \to 0$. We have therefore established that there is a difference when we also maximize vehicle use compared to that of when we only minimize distance.

4.3 Algorithm Implementation

We discuss how the PSO-GA was implemented. We identify the method of encoding each particle or chromosome in the population. Each particle or chromosome is a 'vector' having $2n-1$ dimensions, where n is equivalent to the number of collection sites. In this case, $n = 129$ since we have 129 barangays. We borrow the encoding scheme of Liu et.al. [22] wherein we have n collection sites and a maximum of $n-1$ depots that represent when each vehicle route ends. Instead of using integers, we employ real number like Masrom[24] wherein each particle's component or each chromosome's gene is assigned a real number, specifically we assign a random number uniformly distributed in the interval $(0, 1)$. These numbers will be used to determine the order at which nodes are visited or inserted in the route. This particular encoding scheme is used in order to simplify the methods used in computing particle positions and velocities, and chromosome crossover and mutation. Each particle/chromosome is represented as follows:

$$\begin{array}{cc} \text{Nodes} & \begin{array}{ccccc} v_1 & v_2 & v_3 & ... & v_2n-1 \end{array} \\ \text{Particle} & \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_{2n-1} \end{bmatrix} \end{array}$$

where each r_j, $i = 1, 2, \dots, n$ is a random number uniformly distributed in the interval $(0, 1)$.

For example, if we have 3 collection sites, each particle position or chromosome in the population will have $2(3) - 1 = 5$ components. We let node v_0 to be the depot and nodes v_1 through v_3 as the collection sites. Given

$$\begin{array}{cc} \text{Nodes} & \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \text{Particle} & \begin{bmatrix} 0.3 & 0.4 & 0.1 & 0.2 & 0.5 \end{bmatrix} \end{array}$$

We arrange the nodes based on their respective component values. The nodes higher than v_n are then converted to v_0 to represent that the vehicle returns to the depot and a new vehicle begins its route if there are still unvisited nodes. This results in the sequence,

$$\text{Nodes} \begin{bmatrix} v_3 & v_0 & v_1 & v_2 & v_0 \end{bmatrix}$$

We add zeros to the ends and remove consecutive zeros if necessary. Hence, the sequence of collection becomes:

$$v_0 \to v_3 \to v_0 \to v_1 \to v_2 \to v_0$$

It is important to note that when we use this type of encoding, we do not filter infeasible constructed routes. This problem is solved by the PSO-GA algorithm of H. Garg [14] as discussed in the previous chapters. An initial population of size PSO_{PopNum} is created by generating a set of random particles $PSOx_i, i = 1, 2, \dots, PSO_{PopNum}$. The population will be stored in a matrix, $PSO_Population[PSO_{PopNum}][n]$. Each row is a particle having n dimensions. The initial velocities $PSOv_i, i = 1, 2, \dots, PSO_{PopNum}$ are also randomly produced however, they must follow the maximum and minimum values of velocities $vmax$ and $vmin$ respectively. These bounds are given by the formula

$$\frac{u_j - l_j}{=}\frac{1-0}{=}1, vmax = 1, \text{ and } vmin = -1;$$

where $u_j = 1$ and $l_j = 0$ are the upper and lower bounds of the values that can be taken by the j^{th} component of particle $PSOx_i$, $j = 1, 2, \ldots, n$; $i = 1, 2, \ldots, PSO_{PopNum}$. The fitness value $F(PSOx_i)$ of each particle $PSOx_i$, $i = 1, 2, \ldots, PSO_{PopNum}$ of the population is then obtained by converting the individual into a set of routes using the method presented above. Then we obtain the fitness value of the particle $F(PSOx_i)$ by using the objective function in our model where the input is the set of routes constructed previously. We handle infeasible solutions using the equation

$$F(x_i) = \begin{cases} f(x_i) & \text{if } x_i \text{ is feasible} \\ f_w + \alpha_2 \frac{E}{Q} & \text{otherwise} \end{cases} \tag{4.16}$$

where f_w is the worst fitness value in the current population, E is the total amount of excess demand collected by the vehicle and Q is the maximum capacity of the vehicle. The penalty $\alpha_2 \frac{E}{Q}$ mimics the second term of our objective function (4.4) however, instead of the total amount collected, we use the excess amount of waste collected by the vehicle. The excess amount of waste is equivalent to the demands at collection sites that the vehicle visited and was forced to collect the waste even though it was full.

The initial personal best $Pbest_i$ location of each particle and the global best location $Pbest_g$ of the population is then recorded based from the fitness values obtained.

The new inertia weight value and velocities vectors are then computed using the following equations

$$w = w_{max} - (w_{max} - w_{min}) \times (\frac{PSO_{CurrIt}}{PSO_{MaxIt}}) \tag{4.17}$$

where w_{max} and w_{min} are the upper and lower bounds of the inertia weight. Inertial weight dictates how much of the previous velocity is retained by the individual. PSO_{CurrIt} is the current PSO iteration/generation while PSO_{MaxIt} is the maximum PSO iteration. Note that when the PSO_{CurrIt} becomes equal to the PSO_{MaxIt}, the algorithm stops. We then compute the new velocities and positions of each particle using the following equations.

$$PSOv_i = w \cdot PSOv_i + c_1 \cdot rand() \cdot (Pbest_i - PSOx_i) + c_2 \cdot rand() \cdot (Pbest_g - PSOx_i) \tag{4.18}$$

$$PSOx_i = PSOx_i + PSOv_i \tag{4.19}$$

where c_1 and c_2 are the cognitive and social biases which affect how each particle adapts velocity from personal and global best data. $rand()$ is a random number uniformly distributed in the interval $(0, 1)$. $PSOv_i$ at the left side of the equation (4.18) is the new velocity vector while the one of the right is the old velocity vector. $PSOx_i$ at the left side of equation (4.19) is the new position vector while the one on the right is the old position vector.

The fitness value $F(PSOx_i)$ of each of the particles $PSOx_i$, $i = 1, 2, \ldots, PSO_{PopNum}$ in the new population is obtained. Some members of the new PSO population is then selected to undergo GA where the result of GA replaces an infeasible individual in the population.

The number of particles selected at each PSO iteration is given by GA_{num} which is calculated as:

$$GA_{Num} = GA_{NumMax} - \left(\frac{PSO_{CurrIt}}{PSO_{MaxIt}}\right)^{\gamma} \times (GA_{NumMax} - GA_{NumMin}) \tag{4.20}$$

where GA_{NumMin} and GA_{NumMax} are the minimum and maximum values which GA_{Num} can take. γ is a constant factor that determines how much the ratio of the PSO current and maximum iterations affect the number of individuals obtained. Given that we subtract from the maximum number, it is a given that the number of individuals to be selected becomes lower as the PSO iteration reaches its maximum.

The population size of GA is given by the equation:

$$GA_{PopSize} = GA_{MinPopSize} + \left(\frac{PSO_{CurrIt}}{PSO_{MaxIt}}\right)^{\gamma} \times (GA_{MaxPopSize} - GA_{MinPopSize}) \tag{4.21}$$

where $GA_{MinPopSize}$ and $GA_{MaxPopSize}$ are the minimum and maximum values which $GA_{PopSize}$ can take. γ is the same constant factor above. Given that we add from the minimum number, it is a given that the population size of GA increases at the PSO iteration reaches its maximum.

The maximum iterations of GA is given by the equation:

$$GA_{MaxItr} = GA_{MinItr} + \left(\frac{PSO_{CurrIt}}{PSO_{MaxIt}}\right)^{\beta} \times (GA_{MaxItr} - GA_{MinItr}) \tag{4.22}$$

where GA_{MinItr} and GA_{MaxItr} are the minimum and maximum values which GA_{MaxItr} can take. β is a constant factor that determines how much the ratio of the PSO current and maximum iterations affect the number of individuals obtained. Given that we add from the maximum number, it is a given that the maximum iteration increases as the PSO iteration reaches its maximum.

We set the number of currently selected individuals $GA_{CurrNum} = 0$. We select a random position $PSOx_s$ from the PSO population, s is the index of the selected individual. Then we iterate the number of currently selected individuals by 1, $GA_{CurrNum} = GA_{CurrNum} + 1$.

We generate a random population of size $GA_{PopSize}$, the same method of random generation is used as the one in PSO. Set the first chromosome in the population of GA as the selected position in PSO, $GAx_1 = PSOx_s$. We then set the generation/iteration number of GA to , $GA_{CurrItr} = 1$.

We initialize the replacement chromosome GA_{rep} as the selected particle. However, in later iterations, the replacement chromosome becomes the chromosome in the population that has a better fitness value compared to the current replacement chromosome. This is done in the next step.

We then obtain the fitness values $F(GAx_i)$ of each chromosome $GAx_i, i = 1, 2, \ldots, GAx_{PopSize}$ in the GA population using the same route construction method and objective function in the model. Then we obtain the best feasible solution in the population GAx_b whose fitness value is the minimum, b is the index of the best feasible solution. We then compare the values of the best feasible solution in the GA population and the current replacement chromosome. If the best particle has a better fitness value $(F(GA_{rep}) > F(GAx_b))$ then we replace GA_{rep} with GAx_b. Otherwise, we do not replace it.

We then create the next population by performing roulette wheel selection to obtain pairs of chromosomes GAx_{p1} and GAx_{p2} that would undergo single crossover based on the probability GA_{cross}. Their children will possibly undergo mutation based on the probability GA_{mut}. If crossover occurs, it is done by selecting a random crossover point $cp \in 1, 2, 3, ..., d - 1$ which is the index of the gene where the crossover happens. The first child is made by retaining the genes of first parent, specifically the first until the crossover point cp. Then the remaining genes of the first child is filled in by the genes of the seconds parent from cp to the last gene. The second child is created by same method but we take genes from the seconds parent first before the first parent. We choose the

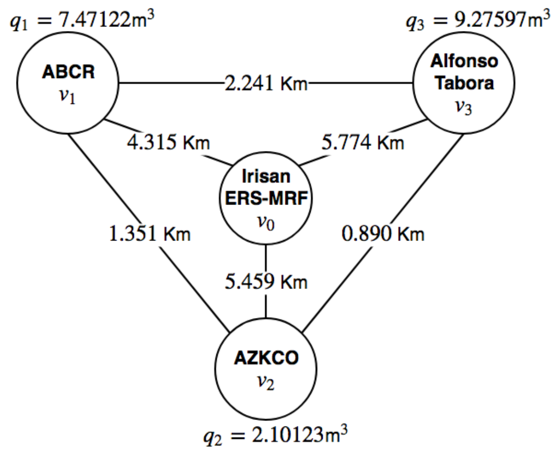

Figure 4.2: Graph of the 4 Nodes Small Scale Example

child that has the better feasible fitness value to be placed in our new population. If the selected child undergoes mutations, some of its genes are re-randomized. After the GA process has terminated, we replace infeasible solutions in the PSO population with GA_{rep}. Note that in total, we replace GA_{num} particles. The PSO process is then looped until either the maximum iterations are reached or the population converges. When the maximum iterations are reached, then we consider that as a failed run. The population is considered 'converged' when the previous and current population is made up of only one solution. This means that the population has become stagnant and that the particles in the population has the same fitness value. If we know the solution to the problem, we can confirm whether or not the converged population is successful or not. If we do not know the solution then we surmise if the solution is successful based on previous runs or results obtained by other studies.

4.4 Algorithm Testing

We now give a small scale example of our main problem. We get the first 3 barangays in the alphabetical list in table A.1 as nodes. Namely, Barangays Andres Bonifacio-Caguioa - Rimando (ABCR), Abanao-Zandueta-Kayong-Chugum-Otek (AZKCO) and Alfonso Tabora. We have a symmetric and complete graph G seen on figure 4.2. The distances are the same from table B.1. The vehicle capacity Q for this case is 12m^3 since this is the capacity of most vehicles used by the SWMD. We have the same $\alpha_1 = 12.474$, $\alpha_2 = 400$ and $\alpha_3 = 840$. We randomized the loads in each barangay as real numbers from 2 to 12 cubic meters. The minimum is set to 2 because most of the vehicles is assigned about 7 to 8 areas to service each day. $\lceil 8/12 \rceil = 2$. The maximum is set to 12 cubic meters because this is stated in assumptions of the model. Also note that even if the were extra waste to be collected in a collection site, a quick response team is used to cover the problem.

The feasible solutions to this instance are as follows

$$\begin{aligned}
v_0 \to v_1 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 = &\quad 12.474 \cdot 31.096 + 1,891.719333 = &\quad \text{Php } 2,279.610837 \\
v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_0 = &\quad 12.474 \cdot 22.673 + 1,051.719333 = &\quad \text{Php } 1,334.542335 \\
v_0 \to v_2 \to v_3 \to v_0 \to v_1 \to v_0 = &\quad 12.474 \cdot 20.753 + 1,051.719333 = &\quad \text{Php } 1,310.592255
\end{aligned}$$

The rest of the solutions violate the capacity constraint.

We test our PSO-GA algorithm using this small scale graph of the problem. The following parameters were used

- PSO Population Size = 5, 10
- PSO Maximum Iterations = PSO Population Size $\times$ 5, " $\times$ 10, " $\times$ 15, " $\times$ 20, " $\times$ 25
- Cognitive and Social factors $c_1 = 1.5$, $c_2 = 1.5$
- Initial and final inertia weight $w_i = 0.9$ $w_f = 0.4$
- Crossover Rate = 0.85
- Mutation Rate = 0.02
- $\gamma = 10$
- $\beta = 15$
- GA Minimum Taken = 1
- GA Maximum Taken = $\lceil$PSO Population Size $\times$ 0.2$\rceil$
- GA Initial Population Size = 10
- GA Final Population Size = 5
- GA Minimum Iterations = 10
- GA Maximum Iterations = 15
- Acceptance Threshold = 1×10^{-5}

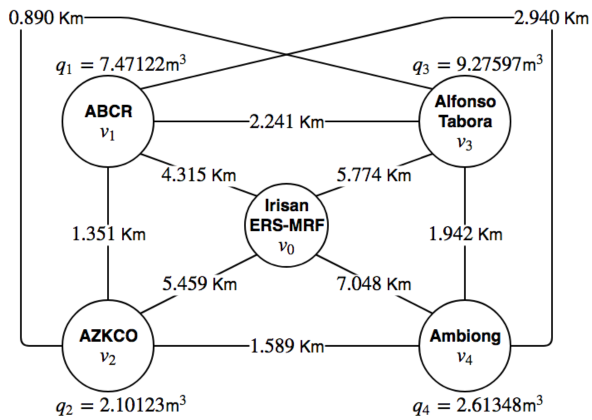

Figure 4.3: Graph of the 5 Nodes Small Scale Example

The algorithm and problems were encoded and run using Matlab v.2015 on a computer with the following specifications:

$$\begin{aligned} \text{CPU} &= \text{Intel i5-6200U 2.3 GHz} \\ \text{RAM} &= \text{16 Gb} \\ \text{OS} &= \text{Windows 10 Home 2017} \end{aligned}$$

We increase the barangays to 4 this time adding Brgy. Ambiong. The graph G is seen on figure 4.3. We also test the algorithm with the same parameters and compare the results from the small scale example with 4 nodes.

The feasible solutions to this instance are as follows

$$\begin{aligned}
v_0 \to v_1 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 \to v_4 \to v_0 = &\quad \text{Php } 3,208.328341 \\
v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_0 \to v_4 \to v_0 = &\quad \text{Php } 2,263.259839 \\
v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 = &\quad \text{Php } 2,237.688139 \\
v_0 \to v_3 \to v_4 \to v_0 \to v_1 \to v_0 \to v_2 \to v_0 = &\quad \text{Php } 2,232.611221 \\
v_0 \to v_2 \to v_3 \to v_0 \to v_1 \to v_0 \to v_4 \to v_0 = &\quad \text{Php } 2,239.309759 \\
v_0 \to v_2 \to v_4 \to v_0 \to v_1 \to v_0 \to v_3 \to v_0 = &\quad \text{Php } 2,232.137209 \\
v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_4 \to v_0 = &\quad \text{Php } 1,287.542719 \\
v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_3 \to v_0 = &\quad \text{Php } 1,294.241257
\end{aligned}$$

We employ the PSO-GA algorithm to the full scale of the problem. The 129 barangays are given in table A.1. All the distances used are seen on table B.1.

Chapter 5

Results and Discussion

We now present the results of the small scale tests. The summary of the results in the small scale test with 4 nodes is seen on table 5.1 while the test with 5 nodes is seen on table 5.2.

Table 5.1: Summary of Results, 4 Node Small Scale Test

Pop. Size	**Max. Iter.**	**Best Value**	**Successful**	**Ave. Run Time (s)**	**Std. Dev.**
5	25	1310.592255	10/10	27.689075	0
	50	1310.592255	9/10	61.706742	0
	75	1310.592255	10/10	30.032919	0
	100	1310.592255	10/10	85.911357	0
	125	1310.592255	10/10	83.594281	0
10	50	1310.592255	7/10	114.393866	0
	100	1310.592255	7/10	189.219563	0
	150	1310.592255	8/10	194.741294	0
	200	1310.592255	7/10	333.625873	0
	250	1310.592255	10/10	244.929403	0

The table above shows that for a small population size of 5, there was no problem with population convergence and obtaining the optimal solution. However, when the population size was doubled, population convergence became a problem. The results showed that on the failed trials, after reaching the maximum iteration, the population was only close to convergence. There were about 2 to 3 individuals in the population that did converge with the population. The best solution was still found but there was not enough iterations for convergence, we see when the maximum iterations was increased to 250, all of the runs converged to the best solution.

Table 5.2: Results of the 5 Node Small Scale Test

Pop. Size	**Max. Iter.**	**Best Value**	**Successful**	**Ave. Run Time (s)**	**Std. Dev.**
5	25	1287.542719	6/10	51.181332	2.118263
	50	1287.542719	7/10	63.927162	3.235701
	75	1287.542719	7/10	89.832843	298.249282
	100	1287.542719	9/10	100.969314	2.118264
	125	1287.542719	8/10	68.577777	398.276007
10	50	1287.542719	8/10	84.009903	0
	100	1287.542719	7/10	187.791896	0
	150	1287.542719	8/10	277.407633	0
	200	1287.542719	7/10	364.537270	0
	250	1287.542719	8/10	292.589868	0

The table above shows almost the same trend as with in table 5.1. For a small population size of 5, as the maximum iterations is gradually increased from 25 to 125, the number of successful convergences more or less increased. The trials that were unsuccessful displayed some convergence on a suboptimal location. This is evidenced by the non-zero standard deviation values. Not all trials resulted in the population obtaining the optimal solution. Some trials converged to the suboptimal solution $v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_3 \to v_0$ instead of the best solution $v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_4 \to v_0$. These two routes have very close cost values, $1,294.241257$ and 1287.542719 respectively. When we inspect the populations per iteration, the runs that converged to the suboptimal solution could not find the optimal solution before it converged. As for the maximum iterations 75, and 125, where there is a large standard deviation, some of the trials converged to the feasible solution, $v_0 \to v_2 \to v_4 \to v_0 \to v_1 \to v_0 \to v_3 \to v_0$. When we inspect the members of the populations per iteration, it is observed that on some of these trials, the population never found better routes for the span of short the iterations. The runs stopped at about 16 to 19 iterations.

For a population size of 10, the convergence rate seemed to have been stagnant from about 70% to 80%. In failed runs, the population did not converge within the maximum iterations set but there were no trials where the population converged to the suboptimal solution. This may be because there were more members that were able explore the solution space as compared to only 5 members where there were trials that did converge on suboptimal routes.

It is observed that in both tests, the PSO-GA algorithm has some consistency to the convergence rate when the population size was doubled from 5.

Chapter 6

Conclusion and Recommendation

A hybrid PSO-GA algorithm was used in order to solve the waste collection vehicle routing problem. The results obtained during the preliminary testing show that the hybrid PSO-GA proposed by Harish Garg[14] can indeed solve the vehicle routing problems however, in order to have a high population convergence rate, the maximum iterations needed must be high enough even for large population sizes. For the next study, it is recommend that the specific collection sites per barangay be used instead of barangay halls and landmarks to have a more accurate model of the problem. It is also recommended that a more dynamic model be used which can reflect the real life situation of waste collection. Dynamism can be employed through adding a time dimension to the problem. Demands can also be set higher than the capacity of a vehicle and partial collection can be employed. It is also recommended that the PSO-GA algorithm of H. Garg[14] be tested under benchmark problems to fully test its efficiency in solving vehicle routing problems.

List of References

[1] M. Akhtar, M. A. Hannan, and H. Basri, *Particle swarm optimization modeling for solid waste collection problem with constraints*, in 2015 IEEE 3rd International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA), Nov 2015, pp. 1–4.

[2] K. Buhrkal, A. Larsen, and S. Ropke, *The waste collection vehicle routing problem with time windows in a city logistics context*, Procedia - Social and Behavioral Sciences, 39 (2012), pp. 241 – 254. Seventh International Conference on City Logistics which was held on June 7- 9,2011, Mallorca, Spain.

[3] J. Carr, *An introduction to genetic algorithms*, (2014).

[4] M. Clerc, *The swarm and the queen: towards a deterministic and adaptive particle swarm optimization*, Evolutionary Computation, (1999).

[5] M. Clerc and J. Kennedy, *The particle swarm - explosion, stability, and convergence in a multidimensional complex space*, IEEE Transactions on Evolutionary Computation, 6 (2002), pp. 58–73.

[6] J.-F. Cordeau, G. Laporte, M. W. Savelsbergh, and D. Vigo, *Chapter 6 vehicle routing*, in Transportation, C. Barnhart and G. Laporte, eds., vol. 14 of Handbooks in Operations Research and Management Science, Elsevier, 2007, pp. 367 – 428.

[7] G. B. Dantzig and J. H. Ramser, *The truck dispatching problem*, Manage. Sci., 6 (1959), pp. 80–91.

[8] K. Deb, *An efficient constraint handling method for genetic algorithms*, Computer Methods in Applied Mechanics and Engineering, 186 (2000), pp. 311–338.

[9] R. C. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, (1995).

[10] ——, *Particle swarm optimization*, (1995).

[11] R. C. Eberhart and Y. Shi, *Particle swarm optimization: developments, applications and resources*, Evolutionary Computation, (2001).

[12] Editors of Encyclopaedia Britannica, *Uncertainty principle*.

[13] Full Advantage Philippines International Incorporated, *Ten-year ecological solid waste management plan 2015-2014*.

[14] H. Garg, *A hybrid pso-ga algorithm for constrained optimization problems*, Applied Mathematics and Computation, 274 (2016), pp. 292–305.

[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 1989.

[16] Q. He and L. Wang, *An effective co-evolutionary particle swarm optimization for constrained engineering design problems*, Engineering Applications of Artificial Intelligence, 20 (2007), pp. 89–99.

[17] S. He, E. Prempauin, and Q. H. Wu, *An improved particle swarm optimizer for mechanical design optimization problems*, Engineering Optimization, 36 (2004), pp. 585–605.

[18] F. Heppner and G. U., *The Ubiquity of Chaos*, AAAS Publications, 1990, ch. A Stochastic Nonlinear Model for Coordinate Bird Flocks.

[19] D. Hoornweg and B. Perinaz, *What a waste: a global review of solid waste management*, Urban Development Series Knowledge Papers, 15 (2012), pp. 87–88.

[20] A. Kaveh and S. Talatahari, *Engineering optimization with hybrid particle swarm and ant colony opitmization*, Asian Journal of Civil Engineering, 10 (2009), pp. 611–628.

[21] H. C. Lacsamana, *Baguio pays out p1 b for waste hauling in 10 yrs*, Baguio Midland Courier, (2017).

[22] J. Liu and J. Lampinen, *A fuzzy adaptive differential evolution algorithm*, Soft Computing, 9 (2005), pp. 448–462.

[23] M. M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research, 35 (1987), pp. 254–266.

[24] S. Masrom, S. Abidin, and A. Nasir, *Hybrid Particle Swarm Optimization for vehicle routing problem with Time Windows*.

[25] Z. Michalewicz and M. Schoenauer, *Evolutionary algorithms for constrained parameter optimization problems*, Evolutionary Computation, 4 (1996), pp. 1–32.

[26] T. Nuortio, J. Kytjoki, H. Niska, and O. Brysy, *Improved route planning and scheduling of waste collection and transport*, Expert Systems with Applications, 30 (2006), pp. 223 – 232.

[27] M. Obitko, *Introduction to genetic algorithms*, 1998.

[28] F. Olowan, E. Bilog, L. Y. Jr., E. Avila, J. Alangsab, E. Datuin, P. Fianza, L. Farinas, A. Allad-iw, B. Bomogao, and M. Lawana, *Baguio city council okays plastic free ordinance*, Sun Star Baguio, (2017).

[29] M. Omran, *Codeq: an effective metaheuristic for continuous global optimisation*, International Journal of Metaheuristics, 1 (2010), pp. 108–131.

[30] C. W. Reynolds, *Flocks, herds, and schools: A distributed behavioral model*, ACM SIGGRAPH Computer Graphics, 21 (1987), pp. 25–34.

[31] D. See, *Approval of citys solid waste plan in order*, Sun Star Baguio, (2017).

[32] ——, *City to purchase 4 more trucks*, Sun Star Baguio, (2018).

[33] L. H. Son, *Optimizing municipal solid waste collection using chaotic particle swarm optimization in gis based environments: A case study at danang city, vietnam*, Expert Systems with Applications, 41 (2014), pp. 8062 – 8074.

[34] J. Sun, B. Feng, and W. Xu, *Particle swarm optimization with particles having quantum behavior*, in Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), vol. 1, June 2004, pp. 325–331 Vol.1.

[35] J. Sun, W. Xu, and B. Feng, *A global search strategy of quantum-behaved particle swarm optimization*, in IEEE Conference on Cybernetics and Intelligent Systems, 2004., vol. 1, Dec 2004, pp. 111–116 vol.1.

[36] D. Tung and A. Pinnoi, *Vehicle routingscheduling for waste collection in hanoi*, European Journal of Operational Research, 125 (2000), pp. 449–468.

[37] T. Xiang, X. Liao, and K.-w. Wong, *An improved particle swarm optimization algorithm combined with piecewise linear chaotic map*, Applied Mathematics and Computation, 190 (2007), pp. 1637–1645.

Appendix A

Table of Node Markers

Table A.1: Nodes and their Google Maps Markers

BARANGAY	**MARKER**
A. Bonifacio-Caguioa-Rimando (ABCR)	Abanao-Zandueta-Kayong-Chugum-Otek (AZKCO), Baguio, Benguet
Abanao-Zandueta-Kayong-Chugum-Otek (AZKCO)	ABCR MULTI-PURPOSE BRGY.HALL, Rimando Road, Baguio City, Benguet, Philippines
Alfonso Tabora	Alfonso Tabora, Baguio City, Benguet, Philippines
Ambiong	AMBIONG BAGUIO BARANGAY HALL, Ambiong Road, Baguio City, Benguet, Philippines
Andres Bonifacio (Lower Bokawkan)	Andres Bonifacio (Lower Bokawkan), Baguio City, Benguet, Philippines
Apugan-Loakan	Apugan Barangay Hall, Loakan Road, Baguio, 2600 Benguet
Asin Road	Asin Road, Baguio City, Benguet, Philippines
Atok Trail	Atok Trail Barangay Hall, Baguio City, Benguet, Philippines
Aurora Hill Proper (Malvar-Sgt. Floresca)	Aurora Hill Proper Barangay Hall, Malvar Street, Baguio City, Benguet, Philippines
Aurora Hill, North Central	Aurora Hill, North Central, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Aurora Hill, South Central	Aurora Hill, South Central, Baguio City, Benguet, Philippines
Bagong Lipunan (Market Area)	Bagong Lipunan (Market Area), Baguio City, Benguet, Philippines
Bakakeng Central	Bakakeng Central, Marcos Highway, Brgy., Baguio, 2600 Benguet
Bakakeng North	Bakakeng Sur Rd, Norte, Baguio, 2600 Benguet
Bal-Marcoville (Marcoville)	Bal-Marcoville (Marcoville), Baguio City, Benguet, Philippines
Balsigan	Balsigan Road, Baguio, Benguet
Bayan Park East	East Bayan Park Aurora Hill Barangay Multi Purpose Hall
Bayan Park Village	Bayan Park Village Barangay Hall
Bayan Park West (Bayan Park)	Bayan Park West (Bayan Park), Baguio City, Benguet
BGH Compound	Baguio General Hospital Driveway, Baguio, 2600 Benguet
Brookside	35 Lower Brookside, Baguio, Benguet
Brookspoint	Brookspoint Rd, Baguio, Benguet
Cabinet Hill-Teacher's Camp	Cabinet Hill-Teacher's Camp, Baguio City, Benguet, Baguio, Benguet

Table A.1 continued from previous page

Camdas Subdivision	Camdas Subdivision, Baguio City, Benguet, Philippines
Camp 7	CAMP 7 BARANGAY HALL, Kennon Road, Baguio City, Benguet, Philippines
Camp 8	Camp 8 Health Center, Kennon Road, Baguio City, Benguet, Philippines
Camp Allen	CAMP HENRY T. ALLEN, Baguio City, Benguet, Philippines
Campo Filipino	CAMPO FILIPINO BARANGAY HALL, Quirino Highway, Baguio City, Benguet, Philippines
City Camp Central	City Camp District Health Center, City Camp Road, City Camp Central, Baguio City, Benguet, Philippines
City Camp Proper	City Camp Barangay Hall, City Camp Road, Baguio City, Benguet, Philippines
Country Club Village	Country Club Village Baguio City, Upper Country Club Road, Baguio, Benguet, Philippines
Cresencia Village	Cresencia Village Barangay, Bado Dangwa, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Dagsian, Lower	83, Lower Dagsian Barangay Hall, Baguio City, Benguet, Philippines
Dagsian, Upper	Upper Dagsian Barangay Hall, Lower Dagsian Road, Baguio City, Benguet, Philippines
Dizon Subdivision	DIZON-MANZANILLO SUBDIVISION, Kalapati Street, Baguio City, Benguet, Philippines
Dominican Hill-Mirador	DOMINICAN - MIRADOR BARANGAY, Extension Road, Baguio City, Benguet, Philippines
Dontogan	Dontogan Barangay Hall, Santo Tomas Road, Baguio City, Benguet, Philippines
DPS Compound	DPS Compound Barangay Hall, DBS Compound, Baguio City, Benguet, Philippines
Engineers' Hill	Engineer's Hill Barangay Hall, Marcoville Street, Baguio City, Benguet, Philippines
Fairview Village	FAIRVIEW Barangay Hall, Upper Fairview Ferguson Road, Baguio City, Benguet, Philippines
Ferdinand (Happy Homes-Campo Sioco)	Brgy. Ferdinand Barangay Hall, Baguio City, Benguet, Philippines
Fort del Pilar	Fort Del Pilar, Loakan Road, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Gabriela Silang	Gabriela Silang Covered Court, Gabriela Silang Road, Baguio City, Benguet, Philippines
General Emilio F. Aguinaldo (QuirinoMagsaysay, Lower)	Danes Bakeshop, Everlasting Street, Baguio
General Luna, Upper	LOWER GENERAL LUNA BARANGAY HALL, Baguio City, Benguet, Philippines
General Luna, Lower	UPPER GENERAL LUNA, Gen. Luna Road, Baguio City, Benguet, Philippines
Gibraltar	Gibraltar Barangay Hall, C. Arellano Street, Baguio City, Benguet, Philippines
Greenwater Village	Greenwater Village, Baguio City, Benguet, Philippines
Guisad Central	Central Guisad Barangay Hall, Pucay Subdivision Road, Baguio City, Benguet, Philippines
Guisad Sorong	Guisad Surong Barangay Hall, Unnamed Road, Baguio City, Benguet, Philippines
Happy Hollow	Happy Hallow Barangay Hall, Brgy. Happy Hallow, Baguio, Benguet
Happy Homes (Happy Homes-Lucban)	Happy Homes (Happy Homes-Lucban) Barangay Hall
Harrison-Claudio Carantes	Harrison-Claudio Carantes, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Hillside	Hillside Baranggay Hall, Hillside Road, Baguio, Benguet
Holy Ghost Extension	Holy Ghost Extension Barangay Hall, Holy Ghost Extension Road, Baguio City, Benguet, Philippines
Holy Ghost Proper	Holy Ghost Proper, Baguio City, Benguet, Philippines
Honeymoon (Honeymoon-Holy Ghost)	Honeymoon-Holyghost Barangay Hall, Baguio City, Benguet, Philippines
Imelda R. Marcos (La Salle)	Imelda R. Marcos (La Salle), Baguio City, Benguet, Philippines
Imelda Village	Imelda Village Barangay Multipurpose Hall, Baguio City, Benguet, Philippines
Irisan	IRISAN BARANGAY HALL, Baguio City, Benguet, Philippines
Kabayanihan	Kabayanihan Barangay Hall, Central Business District, Mabini Street, Baguio, Benguet
Kagitingan	Kagitingan Barangay Hall, Lower Bonifacio Street, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Kayang Extension	Kayang Extension, Baguio City, Benguet, Philippines
Kayang-Hilltop	Kayang Hilltop Barangay, Hilltop Street, Brgy. Kayang Hilltop, Baguio City, Benguet, Philippines
Kias	Kias, Baguio City, Benguet, Philippines
Legarda-Burnham-Kisad	Barangay Office Burnham-Legarda Brgy., Gen. Lim Street, Baguio City, Benguet, Philippines
Liwanag-Loakan	Liwanag-Loakan, Baguio City, Benguet, Philippines
Loakan Proper	Loakan Proper Barangay Hall, Purok Bubon, Baguio City, Benguet, Philippines
Lopez Jaena	Lopez Jaena, Baguio City, Benguet, Philippines
Lourdes Subdivision Extension	Lourdes Subdivision Extension, Baguio City, Benguet, Philippines
Lourdes Subdivision, Lower	Lower Lourdes Day Care and Multipurpose Hall, Baguio City, Benguet, Philippines
Lourdes Subdivision, Proper	Lourdes Barangay Hall, Baguio City, Benguet, Philippines
Lualhati	LUALHATI BARANGAY HALL, Baguio City, Benguet, Philippines
Lucnab	Lucnab, Baguio City, Beanguet, Philippines

Table A.1 continued from previous page

Magsaysay Private Road	Magsaysay Private Road, Baguio City, Benguet, Philippines
Magsaysay, Lower	Lower Magsaysay Barangay Multi-Purpose Hall, Lower Magsaysay Avenue, Baguio City, Benguet, Philippines
Magsaysay, Upper	Magsaysay, Upper, Baguio City, Benguet, Philippines
Malcolm Square-Perfecto (Jose Abad Santos)	Malcolm Square-perfecto (Jose Abad Santos), Baguio City, Benguet, Philippines
Manuel A. Roxas	Manuel Roxas Barangay, Baguio City, Benguet, Philippines
Market Subdivision, Upper	Market Subdivision, Upper, Baguio City, Benguet, Philippines
Middle Quezon Hill Subdivision (Quezon Hill Middle)	Middle Quezon Hill Subdivision (Quezon Hill M, Baguio City, Benguet, Philippines
Military Cut-off	MILITARY CUT OFF BARANGAY HALL, Military Cutoff Road, Baguio City, Benguet, Philippines
Mines View Park	Mines View Multipurpose Cooperative, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Modern Site, East	EAST MODERNSITE BRGY, P. Ledesma Street, Baguio, Benguet
Modern Site, West	Modern Site, West, Baguio City, Benguet, Philippines
MRR-Queen of Peace	MRR-Queen Of Peace, Baguio City, Benguet, Philippines
New Lucban	New Lucban Barangay Hall, New Lucban Road, Baguio City, Benguet, Philippines
Outlook Drive	Outlook Drive South, Baguio City, Benguet, Philippines
Pacdal	Pacdal Barangay Hall, Siapno Road, Baguio City, Benguet, Philippines
Padre Burgos	P. BURGOS MULTI PURPOSE HALL, Upper P. Burgos, Baguio, Benguet
Padre Zamora	Padre Zamora Barangay Hall
Palma-Urbano (Cario-Palma)	Palma-Urbano (Carino-Palma), Baguio City, Benguet, Philippines
Phil-Am	Barangay Hall Phil-am, Worcester Road, Baguio City, Benguet, Philippines
Pinget	Pinget Barangay Hall, Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Pinsao Pilot Project	Barangay Hall, Pinsao Road, Baguio City, Benguet, Philippines
Pinsao Proper	Pinsao Proper Barangay Hall, Baguio City, Benguet, Philippines
Poliwes	POLIWES BARANGAY HALL, Puliwes Road, Baguio City, Benguet, Philippines
Pucsusan	Pucsusan, Baguio City, Benguet, Philippines
Quezon Hill Proper	Quezon Hill Proper Barangay Hall, Quezon Hill Road 1, Baguio City, Benguet, Philippines
Quezon Hill, Upper	Upper Quezon Hill, Tin, Brgy Upper Quezon Hill, Baguio, Benguet
Quirino Hill, East	Block 7, East Quirino Hill Barangay Hall, Quirino Hill Road, Baguio, Benguet
Quirino Hill, Lower	Lower Quirino Hill Barangay Hall, Baguio, Benguet
Quirino Hill, Middle	MIDDLE QUIRINO HILL BARANGAY HALL, Baguio, Benguet
Quirino Hill, West	West Quirino Hill Barangay Hall, Baguio, Benguet
Quirino-Magsaysay, Upper (Upper QM)	Upper Q - M Barangay Hall, Jasmin Street, Barangay Upper Q.M., Baguio City, Benguet, Philippines

Table A.1 continued from previous page

Rizal Monument Area	Rizal Monument Barangay Hall, Baguio, Benguet
Rock Quarry, Lower	MULTI PURPOSE BRGY. HALL LOWER ROCK QUARRY, Lower Rock Quarry, Baguio City, Benguet, Philippines
Rock Quarry, Middle	Barangay Middle Rock Quarry Multi - Purpose Building, Lower Rock Quarry, Brgy. Middle Rock Quarry, Baguio, Benguet
Rock Quarry, Upper	Upper Rock Quarry Barangay Hall, Lower Rock Quarry, Brgy. Upper Rock Quarry, Baguio City, Benguet, Philippines
Saint Joseph Village	St. Joseph Village Barangay Hall, Everlasting, Navy Base - Polo Field, St. Joseph Village, Baguio, Benguet
Salud Mitra	Barangay Hall, Baguio City, Benguet, Philippines
San Antonio Village	Leonila Hill Barangay Hall, Evangelista Street, Baguio City, Benguet, Philippines
San Luis Village	SAN LUIS VILLAGE BARANGAY HALL, Asin Road, Baguio City, Benguet, Philippines
San Roque Village	Church of Christ at Pines, Baguio, Benguet
San Vicente	San Vicente-Baguio City Multipurpose Cooperative, Kennon Road, Baguio, Benguet

Table A.1 continued from previous page

Sanitary Camp, North	Sanitary Camp, North, Baguio City, Benguet, Philippines
Sanitary Camp, South	Brgy. South Sanitary Camp Multi Purpose Hall, South Sanitary Camp Road, Baguio City, Benguet, Philippines
Santa Escolastica	Santa Escolastica Village Hall, Sta. Escolastica, Baguio, Benguet
Santo Rosario	Santo Rosario Barangay Hall, Sto. Rosario Village Road, Baguio City, Benguet, Philippines
Santo Tomas Proper	Sto Tomas Proper Barangay Hall, Baguio, Benguet
Santo Tomas School Area	Santo Tomas School Area, Baguio City, Benguet, Philippines
Scout Barrio	Scout Barrio Basketball Court, Baguio City, Benguet, Philippines
Session Road Area	BARANGAY HALL, Gov. Pack Road, Baguio City, Benguet, Philippines
Slaughter House Area (Santo Nio Slaughter)	BARANGAY STO. NIO SLAUGHTER BARANGAY HALL

Table A.1 continued from previous page

SLU-SVP Housing Village	SLU-SVP Housing Village Barangay, Baguio City, Benguet, Philippines
South Drive	Southdrive Barangay, South Drive, Baguio City, Benguet, Philippines
Teodora Alonzo	T Alonzo Barangay Hall, T. Alonzo Street, Baguio City, Benguet, Philippines
Trancoville	Trancoville, Baguio City, Benguet, Philippines
Victoria Village	VICTORIA VILLAGE BARANGAY HALL, Baguio City, Benguet, Philippines
DEPOT	**MARKER**
Irisan Dumpsite	Purok 18, Barangay Irisan, Baguio, 2600 Benguet

Appendix B

Table Showing the Distances Between each Barangay Marker

Table B.1: Distances between Nodes in Kilometers

NODES	depot	1	2	3	4	5	6	7	8
depot	0	4.315	5.459	5.774	7.048	4.13	11.155	3.42	11.944
1	4.315	0	1.351	2.241	2.94	1.396	7.537	5.443	8.326
2	5.459	1.351	0	0.89	1.589	2.474	7.93	6.489	8.703
3	5.774	2.241	0.89	0	1.942	2.09	8.82	6.449	9.593
4	7.048	2.94	1.589	1.942	0	3.87	9.519	8.078	10.292
5	4.13	1.396	2.474	2.09	3.87	0	8.456	4.975	9.245
6	11.155	7.537	7.93	8.82	9.519	8.456	0	12.21	0.789
7	3.42	5.443	6.489	6.449	8.078	4.975	12.21	0	12.999
8	11.944	8.326	8.703	9.593	10.292	9.245	0.789	12.999	0
9	6.234	2.126	0.775	1.057	0.888	3.054	8.701	7.264	9.474
10	6.537	2.43	1.079	1.327	0.632	3.255	9.004	7.568	9.777
11	6.593	2.883	1.534	1.491	1.45	3.539	8.9	7.671	9.657
12	4.799	0.709	1.136	1.942	2.725	1.338	7.722	5.401	8.511
13	6.044	3.343	4.334	5.182	5.923	4.108	7.291	5.43	8.079
14	6.437	3.286	4.336	5.145	5.925	4.133	5.322	7.251	6.11
15	6.117	2.013	2.398	3.288	3.845	2.861	5.807	6.924	6.596
16	5.774	2.461	3.451	4.256	5.04	3.226	5.592	6.753	6.38
17	7.179	3.071	1.72	2.073	0.173	4.001	9.646	8.209	10.419
18	6.829	2.721	1.37	1.723	0.219	3.651	9.3	7.859	10.073
19	6.657	3.357	2.136	1.246	1.251	3.214	10.066	7.692	10.839

Table B.1 continued from previous page

20	5.418	1.932	2.706	3.596	4.295	2.886	6.171	6.473	6.96
21	5.945	1.837	0.486	1.136	1.619	2.96	8.416	6.975	9.189
22	7.118	3.011	1.66	2.016	0.459	3.956	9.585	8.149	10.358
23	6.894	2.683	2.329	2.955	2.884	3.678	7.627	7.741	8.313
24	5.643	2.087	1.528	0.638	2.356	1.904	8.981	6.245	9.676
25	8.336	4.759	5.629	6.519	7.218	5.712	2.993	9.352	3.781
26	5.905	2.322	3.192	4.082	4.781	3.276	5.43	6.966	6.218
27	4.35	0.577	1.585	1.901	3.174	0.889	7.637	4.952	8.426
28	3.986	0.888	1.949	1.861	3.538	0.848	7.931	4.588	8.72
29	4.132	0.786	2.006	2.598	3.595	1.586	7.553	4.657	8.342
30	4.058	0.68	1.9	2.492	3.489	1.48	7.447	4.763	8.236
31	8.829	4.618	4.38	4.919	4.969	5.626	8.383	9.689	9.172
32	4.566	1.374	2.059	1.336	3.116	0.754	8.434	5.411	9.223
33	7.536	3.918	4.109	4.999	5.698	4.851	4.746	8.591	5.535
34	7.387	3.769	4.162	5.052	5.751	4.688	4.17	8.442	4.959
35	5.414	1.859	1.436	0.546	2.298	1.682	8.753	5.983	9.448
36	3.832	1.964	3.154	3.066	4.743	1.593	8.731	4.4	9.52
37	6.616	5.832	6.822	7.679	8.411	6.667	9.587	6.002	10.375
38	6.269	2.165	2.55	3.44	3.997	3.013	5.687	7.076	6.476
39	5.837	1.878	1.979	2.869	3.551	2.376	6.214	6.439	6.9
40	3.582	1.626	2.705	2.321	4.101	0.548	8.686	4.742	9.475
41	5.086	1.982	3.167	3.794	4.756	2.782	6.28	6.065	7.068
42	14.585	10.967	11.36	12.25	12.949	11.886	3.43	15.64	3.679
43	7.451	3.846	4.037	4.927	5.626	4.766	5.049	8.506	5.821
44	3.941	0.837	2.057	2.649	3.646	1.637	7.311	4.92	8.1
45	5.094	0.884	0.978	1.868	2.567	1.976	6.952	6.039	7.725
46	5.463	1.252	1.281	2.171	2.87	2.345	6.915	6.408	7.601
47	8.31	4.121	3.941	4.651	4.409	5.188	8.88	9.251	9.669
48	6.969	2.89	3.149	4.039	4.738	3.671	4.999	7.734	5.788

Table B.1 continued from previous page

49	5.109	2.399	2.406	1.626	3.484	1.058	9.196	6.033	9.985
50	4.591	2.097	2.78	1.982	3.762	0.943	9.081	5.6	9.87
51	10.606	6.395	6.357	6.896	6.682	7.46	10.217	11.523	11.005
52	5.556	1.967	1.258	0.368	2.226	1.79	8.853	6.229	9.556
53	4.854	0.626	1.394	2.284	2.983	1.935	7.083	5.697	7.872
54	6.88	3.262	3.453	4.343	5.042	4.195	4.477	7.935	5.25
55	6.356	2.248	1.21	2.1	2.52	3.371	8.044	7.386	8.833
56	5.777	1.567	1.301	2.191	2.89	2.659	7.579	6.722	8.265
57	5.636	1.447	1.548	2.438	3.137	2.552	6.546	6.615	7.331
58	5.803	2.652	3.643	4.491	5.232	3.487	6.6	6.121	7.388
59	6.831	2.723	1.49	2.115	2.045	3.846	7.607	7.861	8.358
60	1.424	5.683	6.776	6.688	8.365	5.215	12.45	4.228	13.239
61	4.809	0.62	1.263	2.153	2.852	1.693	7.237	5.756	8.01
62	4.861	0.672	0.679	1.569	2.268	1.984	7.566	5.984	8.339
63	4.534	0.779	1.792	2.103	3.381	1.073	7.517	5.136	8.203
64	4.668	0.508	1.337	2.143	2.926	1.207	7.521	5.27	8.31
65	12.806	9.188	9.565	10.455	11.154	10.107	1.651	13.861	1.9
66	4.888	1.022	2.206	3.03	3.795	2	6.919	5.943	7.708
67	10.015	6.438	7.029	7.919	8.618	7.321	1.889	11.031	2.678
68	9.685	6.073	6.978	7.868	8.567	6.991	1.644	10.701	2.432
69	6.632	2.524	1.174	1.318	0.727	3.228	9.099	7.663	9.872
70	3.773	1.548	2.735	3.007	4.324	1.534	8.188	4.341	8.977
71	3.529	1.653	2.86	3.002	4.449	1.529	8.167	4.336	8.956
72	3.548	1.807	2.87	2.782	4.459	1.309	8.517	4.116	9.306
73	8.342	4.153	4.115	4.765	4.741	5.093	9.333	9.156	10.051
74	10.351	6.152	5.89	6.51	6.439	7.218	10.204	11.281	10.992
75	5.32	1.682	1.708	1.558	2.91	2.132	8.576	6.577	9.349
76	5	1.164	1.19	1.127	2.779	1.722	8.058	6.207	8.831
77	5.677	1.488	1.495	2.385	2.572	2.063	7.926	6.401	8.715

Table B.1 continued from previous page

78	4.711	0.522	0.985	1.875	2.574	1.831	7.471	5.834	8.256
79	6.927	3.144	1.911	2.531	2.46	3.466	8.028	7.529	8.779
80	5.551	1.362	1.369	2.259	2.699	1.937	8.053	6.275	8.842
81	3.56	2.779	3.809	3.682	5.398	1.91	9.607	4.986	10.396
82	5.505	1.887	2.792	3.682	4.381	2.806	5.65	6.56	6.439
83	9.021	4.878	4.616	5.236	5.165	5.944	9.636	10.007	10.424
84	6.598	2.49	1.14	1.496	0.986	3.475	9.069	7.629	9.842
85	6.047	1.939	0.588	1.126	1.158	3.062	8.518	7.077	9.291
86	4.037	1.812	2.999	3.271	4.588	1.798	8.548	4.605	9.337
87	5.107	1.493	0.489	1.379	2.078	2.122	7.963	6.185	8.736
88	8.59	4.379	4.407	5.106	5.144	5.374	9.035	9.437	9.824
89	7.361	3.15	3.178	3.86	4.049	4.145	7.958	8.208	8.747
90	4.982	1.373	1.38	1.656	2.969	1.283	8.145	5.584	8.934
91	5.343	1.154	1.161	2.051	2.75	2.004	7.891	6.067	8.664
92	3.689	0.626	1.856	2.592	3.445	1.58	7.466	4.945	8.255
93	5.231	2.069	3.06	3.95	4.649	3.047	6.484	6.369	7.273
94	5.808	2.941	2.518	1.628	3.38	2.724	9.741	7.065	10.53
95	5.572	3.202	3.279	2.642	4.053	2.284	10.152	6.998	10.941
96	3.734	2.862	3.547	2.824	4.604	1.708	9.781	5.16	10.57
97	6.446	2.869	3.739	4.629	5.328	3.823	5.139	7.462	5.927
98	8.665	4.864	4.861	5.571	5.329	5.859	9.672	9.922	10.461
99	3.278	2.063	3.141	2.896	4.676	1.124	9.123	4.704	9.912
100	3.274	2.554	3.641	3.396	5.176	1.624	9.321	4.999	10.11
101	6.186	3.285	2.773	1.883	3.554	3.142	10.117	7.49	10.906
102	5.35	2.449	1.944	1.086	2.718	2.306	9.249	6.693	10.038
103	5.628	2.727	2.168	1.278	2.996	2.584	9.527	6.885	10.316
104	6.48	3.579	3.02	2.13	3.848	3.436	10.381	7.739	11.17
105	4.518	1.414	2.634	3.218	4.223	2.214	6.996	5.497	7.784
106	4.145	0.354	1.395	2.285	2.984	1.308	7.304	5.268	8.093

Table B.1 continued from previous page

107	3.658	1.08	2.3	2.884	3.889	1.88	7.594	4.909	8.383
108	3.307	1.431	2.651	3.187	4.24	1.714	7.945	4.558	8.734
109	3.566	1.172	2.392	2.976	3.981	1.972	7.686	4.817	8.475
110	6.84	3.039	3.067	3.639	3.828	4.034	7.847	8.097	8.636
111	8.336	4.759	5.629	6.519	7.218	5.712	2.993	9.352	3.781
112	6.843	2.9	1.549	1.433	0.509	3.361	9.475	7.702	10.248
113	3.881	2.661	3.755	3.667	5.344	2.193	9.428	2.81	10.217
114	2.543	6.802	7.895	7.807	9.484	6.334	13.569	5.347	14.358
115	6.29	2.978	3.967	4.773	5.556	3.743	5.443	7.269	6.231
116	6.475	3.574	2.353	1.463	1.467	3.391	10.283	7.732	11.056
117	5.393	2.492	1.161	0.381	1.959	2.309	9.087	6.65	9.86
118	6.669	3.086	3.779	4.453	5.368	4.04	4.749	7.73	5.538
119	5.886	2.782	3.967	4.594	5.556	3.582	7.08	6.149	7.868
120	5.844	5.748	6.739	7.596	8.328	6.584	6.962	5.919	7.75
121	10.158	10.063	11.053	11.91	12.642	10.898	13.818	10.233	14.606
122	7.471	3.858	4.251	5.141	5.84	4.777	3.854	8.531	4.643
123	5.618	1.429	1.53	2.42	3.119	2.447	6.773	6.51	7.562
124	4.829	1.335	1.361	1.067	2.902	1.785	8.1	6.086	8.873
125	6.233	3.41	4.594	5.269	6.183	4.257	5.678	7.375	6.466
126	7.78	3.578	3.66	4.286	4.215	4.459	6.747	8.522	7.536
127	4.983	1.369	0.476	1.366	2.065	1.998	7.95	6.061	8.723
128	5.577	1.954	0.603	0.496	1.561	2.493	8.529	6.834	9.302
129	2.819	2.038	3.116	3.028	4.705	1.311	8.866	4.245	9.655
NODES	9	10	11	12	13	14	15	16	17
depot	6.234	6.537	6.593	4.799	6.044	6.437	6.117	5.774	7.179
1	2.126	2.43	2.883	0.709	3.343	3.286	2.013	2.461	3.071
2	0.775	1.079	1.534	1.136	4.334	4.336	2.398	3.451	1.72
3	1.057	1.327	1.491	1.942	5.182	5.145	3.288	4.256	2.073

Table B.1 continued from previous page

4	0.888	0.632	1.45	2.725	5.923	5.925	3.845	5.04	0.173
5	3.054	3.255	3.539	1.338	4.108	4.133	2.861	3.226	4.001
6	8.701	9.004	8.9	7.722	7.291	5.322	5.807	5.592	9.646
7	7.264	7.568	7.671	5.401	5.43	7.251	6.924	6.753	8.209
8	9.474	9.777	9.657	8.511	8.079	6.11	6.596	6.38	10.419
9	0	0.378	0.916	1.911	5.109	5.111	3.03	4.226	1.019
10	0.378	0	0.952	2.214	5.412	5.414	3.328	4.529	0.763
11	0.916	0.952	0	2.27	5.468	5.47	3.093	4.585	1.581
12	1.911	2.214	2.27	0	3.738	3.693	2.026	2.856	2.856
13	5.109	5.412	5.468	3.738	0	1.969	3.264	1.854	6.054
14	5.111	5.414	5.47	3.693	1.969	0	2.88	1.797	6.056
15	3.03	3.328	3.093	2.026	3.264	2.88	0	2.381	3.976
16	4.226	4.529	4.585	2.856	1.854	1.797	2.381	0	5.171
17	1.019	0.763	1.581	2.856	6.054	6.056	3.976	5.171	0
18	0.669	0.413	1.231	2.506	5.704	5.706	3.626	4.821	0.35
19	1.59	1.291	1.683	3.105	6.204	6.308	4.534	5.322	1.382
20	3.481	3.784	3.84	2.349	2.169	2.171	1.636	1.139	4.426
21	0.888	1.192	1.048	1.622	4.82	4.822	2.804	3.937	1.792
22	0.959	0.701	1.533	2.795	5.993	5.995	3.915	5.11	0.59
23	2.161	2.373	2.23	2.796	4.841	4.525	1.898	3.7	3.015
24	1.54	1.871	2.019	1.855	4.934	4.889	3.404	4.052	2.487
25	6.404	6.707	6.763	5.27	4.298	2.329	4.194	2.599	7.349
26	3.967	4.27	4.326	2.836	2.56	2.451	1.757	0.924	4.912
27	2.313	2.616	2.719	0.449	3.289	3.244	2.02	2.407	3.305
28	2.677	2.98	3.083	0.813	3.583	3.538	2.336	2.701	3.669
29	2.752	3.084	3.14	1.146	2.805	2.759	2.583	2.096	3.726
30	2.646	2.978	3.034	1.04	2.699	2.653	2.477	1.99	3.62
31	4.505	4.337	4.102	4.744	6.078	5.997	3.365	5.171	5.1
32	2.3	2.501	2.785	1.316	4.086	4.041	2.839	3.274	3.247

Table B.1 continued from previous page

33	4.88	5.183	5.402	4.225	4.153	4.082	2.312	2.724	5.825
34	4.933	5.236	5.277	4.165	3.99	3.933	2.25	2.585	5.878
35	1.482	1.683	1.995	1.741	4.672	4.627	3.176	3.79	2.429
36	3.882	4.185	4.288	2.018	3.983	3.937	3.541	3.274	4.874
37	7.597	7.9	7.956	6.227	2.637	4.458	5.752	4.343	8.542
38	3.176	3.48	3.245	2.178	3.358	3.032	0.152	2.475	4.122
39	2.736	3.04	2.805	2.006	3.548	3.112	0.485	2.287	3.682
40	3.285	3.486	3.77	1.569	4.338	4.293	3.139	3.456	4.232
41	3.942	4.245	4.301	2.342	1.408	1.351	2.097	0.688	4.887
42	12.131	12.434	12.292	11.151	10.72	8.751	9.237	9.021	13.076
43	4.795	5.098	5.317	4.153	4.068	3.997	2.227	2.639	5.753
44	2.803	3.106	3.191	1.197	2.542	2.496	2.49	1.833	3.777
45	1.753	2.056	2.512	1.094	3.36	3.362	1.555	2.477	2.698
46	2.056	2.359	2.815	1.463	3.473	3.475	1.186	2.59	3.001
47	3.594	3.898	3.663	4.306	6.343	5.931	3.194	5.106	4.582
48	3.924	4.227	4.046	2.934	3.572	3.515	1.019	2.234	4.869
49	2.683	2.859	3.075	2.396	4.848	4.951	3.919	4.179	3.615
50	2.946	3.147	3.431	2.039	4.733	4.764	3.486	3.997	3.893
51	5.867	6.171	6.079	6.578	7.912	7.831	5.199	7.005	6.855
52	1.425	1.601	1.749	1.642	4.814	4.845	3.216	4.008	2.357
53	2.132	2.435	2.717	1.335	3.078	3.318	1.387	2.493	3.114
54	4.224	4.527	4.746	3.569	3.497	3.44	1.656	2.068	5.169
55	1.705	2.009	1.774	2.033	5.231	5.155	2.275	4.348	2.651
56	2.076	2.379	2.835	1.777	4.039	4.041	1.85	3.156	3.021
57	2.323	2.627	2.937	2.112	3.54	3.39	0.85	2.565	3.268
58	4.418	4.721	4.777	3.047	0.839	1.278	2.573	1.163	5.363
59	1.23	1.534	1.299	2.508	5.064	4.68	1.8	4.181	2.176
60	7.504	7.807	7.91	5.64	6.851	7.656	7.163	6.993	8.496
61	2.038	2.341	2.797	1.253	3.645	3.647	1.709	2.762	2.983

Table B.1 continued from previous page

62	1.454	1.758	2.213	1.148	3.793	3.829	1.989	3.004	2.399
63	2.567	2.87	2.973	0.703	3.426	3.38	1.788	2.661	3.512
64	2.112	2.415	2.471	0.201	3.607	3.562	1.825	2.725	3.057
65	10.336	10.639	10.513	9.373	8.941	6.972	7.458	7.242	11.281
66	2.981	3.284	3.378	1.602	2.321	2.264	2.281	1.439	3.926
67	7.8	8.103	7.976	6.836	5.977	4.008	4.94	4.278	8.745
68	7.753	8.056	8.112	6.592	5.647	3.678	5.543	3.948	8.698
69	0.473	0.095	1.047	2.309	5.507	5.509	3.423	4.624	0.858
70	3.51	3.814	3.908	1.905	3.841	3.795	3.33	3.132	4.455
71	3.619	3.939	4.007	1.954	3.398	3.352	3.259	2.689	4.58
72	3.598	3.901	4.004	1.734	3.748	3.702	3.257	3.039	4.59
73	3.926	4.23	3.995	4.087	6.535	6.263	3.526	5.438	4.872
74	5.624	5.928	5.693	6.336	7.875	7.818	5.186	6.992	6.57
75	2.096	2.4	2.598	1.366	4.803	4.855	2.94	3.97	3.041
76	1.965	2.264	2.55	1.046	4.285	4.321	2.481	3.496	2.91
77	1.758	2.062	2.442	1	4.384	4.327	2.574	3.501	2.703
78	1.76	2.064	2.519	1.231	3.645	3.588	1.775	2.763	2.705
79	1.645	1.949	1.714	2.929	4.882	4.825	2.221	3.999	2.591
80	1.885	2.189	2.503	0.874	4.361	4.363	2.448	3.478	2.83
81	4.584	4.847	4.991	2.721	4.859	4.813	4.244	4.15	5.529
82	3.567	3.87	3.926	2.407	2.108	2.051	1.36	1.225	4.512
83	4.35	4.654	4.419	5.062	7.099	6.687	3.95	5.862	5.296
84	0.439	0.475	0.477	2.275	5.473	5.475	3.395	4.59	1.117
85	0.344	0.648	1.209	1.724	4.922	4.924	2.986	4.039	1.289
86	3.774	4.078	4.172	2.169	4.105	4.059	3.594	3.396	4.719
87	1.254	1.567	1.734	0.784	4.272	4.274	2.29	3.389	2.209
88	4.329	4.633	4.398	4.492	6.6	6.204	3.577	5.379	5.317
89	3.476	3.688	3.545	3.263	5.371	4.975	2.151	4.15	4.18
90	2.155	2.458	2.514	0.885	4.273	4.228	2.459	3.391	3.1

Table B.1 continued from previous page

91	1.936	2.239	2.295	0.666	4.153	4.155	2.24	3.27	2.881
92	2.631	2.934	2.99	1.141	3.098	3.052	2.433	2.216	3.576
93	3.835	4.138	4.194	2.609	1.824	1.826	1.663	0.942	4.78
94	2.564	2.765	3.008	2.481	5.754	5.709	4.055	4.872	3.511
95	3.35	3.727	3.783	3.187	6.017	5.961	4.761	5.135	4.184
96	3.788	3.989	4.273	2.804	5.033	4.987	4.327	4.324	4.735
97	4.514	4.817	4.873	3.353	2.563	2.506	2.304	0.709	5.459
98	4.514	4.818	4.583	4.977	7.085	6.689	3.865	5.864	5.502
99	3.844	4.061	4.275	2.005	4.577	4.531	3.528	3.868	4.807
100	4.344	4.561	4.775	2.505	4.573	4.527	4.028	3.864	5.307
101	2.738	2.939	3.264	2.872	6.179	6.173	4.446	5.297	3.685
102	1.902	2.103	2.448	2.217	5.382	5.337	3.766	4.5	2.849
103	2.212	2.413	2.659	2.267	5.574	5.576	3.841	4.692	3.127
104	3.032	3.233	3.511	3.121	6.428	6.43	4.695	5.546	3.979
105	3.38	3.683	3.768	1.774	2.124	2.067	2.494	1.404	4.354
106	2.17	2.473	2.529	0.753	3.17	3.113	1.972	2.288	3.115
107	3.046	3.378	3.434	1.44	2.825	2.779	2.773	2.116	4.02
108	3.397	3.729	3.785	1.791	3.176	3.13	3.037	2.467	4.371
109	3.138	3.47	3.526	1.532	2.917	2.871	2.778	2.208	4.112
110	3.365	3.577	3.434	3.152	5.26	4.864	2.04	4.039	3.959
111	6.404	6.707	6.763	5.27	4.298	2.329	4.194	2.599	7.349
112	0.848	0.592	0.941	2.459	5.883	5.885	3.799	5	0.64
113	4.483	4.786	4.889	2.619	4.68	4.634	4.142	3.971	5.475
114	8.623	8.926	9.029	6.759	7.97	8.775	8.282	8.112	9.615
115	4.742	5.045	5.101	3.373	2.37	2.314	2.532	0.517	5.687
116	1.805	1.508	1.898	3.114	6.421	6.376	4.751	5.539	1.598
117	1.328	1.334	1.761	2.032	5.339	5.294	3.414	4.457	2.09
118	4.554	4.857	4.664	3.486	3.324	3.215	1.571	1.604	5.499
119	4.742	5.045	5.101	3.142	0.867	1.808	2.578	1.488	5.687

Table B.1 continued from previous page

120	7.514	7.817	7.873	6.144	2.553	3.671	5.342	4.259	8.459
121	11.828	12.131	12.187	10.458	6.868	8.656	9.657	8.574	12.773
122	5.022	5.325	5.22	4.042	4.079	4.022	2.127	2.663	5.967
123	2.305	2.608	2.664	1.565	2.804	2.806	1.087	1.921	3.25
124	2.014	2.3	2.504	0.875	4.362	4.364	2.449	3.479	3.033
125	5.369	5.672	5.766	3.817	2.093	0.356	3.004	1.921	6.314
126	3.492	3.704	3.561	3.591	4.389	4.332	1.676	3.506	4.346
127	1.251	1.554	1.61	0.66	4.148	4.15	2.277	3.265	2.196
128	0.674	1.051	1.107	1.513	4.937	4.939	2.856	4.054	1.692
129	3.844	4.147	4.25	1.98	4.118	4.072	3.503	3.409	4.836
NODES	18	19	20	21	22	23	24	25	26
depot	6.829	6.657	5.418	5.945	7.118	6.894	5.643	8.336	5.905
1	2.721	3.357	1.932	1.837	3.011	2.683	2.087	4.759	2.322
2	1.37	2.136	2.706	0.486	1.66	2.329	1.528	5.629	3.192
3	1.723	1.246	3.596	1.136	2.016	2.955	0.638	6.519	4.082
4	0.219	1.251	4.295	1.619	0.459	2.884	2.356	7.218	4.781
5	3.651	3.214	2.886	2.96	3.956	3.678	1.904	5.712	3.276
6	9.3	10.066	6.171	8.416	9.585	7.627	8.981	2.993	5.43
7	7.859	7.692	6.473	6.975	8.149	7.741	6.245	9.352	6.966
8	10.073	10.839	6.96	9.189	10.358	8.313	9.676	3.781	6.218
9	0.669	1.59	3.481	0.888	0.959	2.161	1.54	6.404	3.967
10	0.413	1.291	3.784	1.192	0.701	2.373	1.871	6.707	4.27
11	1.231	1.683	3.84	1.048	1.533	2.23	2.019	6.763	4.326
12	2.506	3.105	2.349	1.622	2.795	2.796	1.855	5.27	2.836
13	5.704	6.204	2.169	4.82	5.993	4.841	4.934	4.298	2.56
14	5.706	6.308	2.171	4.822	5.995	4.525	4.889	2.329	2.451
15	3.626	4.534	1.636	2.804	3.915	1.898	3.404	4.194	1.757
16	4.821	5.322	1.139	3.937	5.11	3.7	4.052	2.599	0.924

Table B.1 continued from previous page

17	0.35	1.382	4.426	1.792	0.59	3.015	2.487	7.349	4.912
18	0	1.032	4.076	1.483	0.528	2.757	2.137	6.999	4.562
19	1.032	0	4.842	2.346	1.56	3.586	1.801	7.765	5.328
20	4.076	4.842	0	3.192	4.365	3.276	3.545	3.189	0.753
21	1.483	2.346	3.192	0	1.714	1.843	1.774	6.115	3.678
22	0.528	1.56	4.365	1.714	0	3.046	2.498	7.288	4.851
23	2.757	3.586	3.276	1.843	3.046	0	3.575	5.839	3.402
24	2.137	1.801	3.545	1.774	2.498	3.575	0	6.468	4.032
25	6.999	7.765	3.189	6.115	7.288	5.839	6.468	0	2.437
26	4.562	5.328	0.753	3.678	4.851	3.402	4.032	2.437	0
27	2.955	3.064	1.998	2.071	3.197	2.79	1.871	4.823	2.388
28	3.319	3.107	2.194	2.435	3.561	3.153	1.657	5.117	2.751
29	3.376	3.761	1.816	2.492	3.636	3.36	2.342	4.695	2.309
30	3.27	3.655	1.71	2.386	3.53	3.254	2.236	4.589	2.203
31	4.75	5.55	4.323	4.187	4.918	2.344	5.557	6.891	4.454
32	2.897	2.499	2.697	2.337	3.202	3.657	1.15	5.62	3.254
33	5.479	6.245	2.399	4.595	5.764	4.01	5.16	4.237	1.884
34	5.532	6.298	2.236	4.648	5.817	3.939	5.213	4.098	1.745
35	2.079	1.709	3.283	1.547	2.384	3.39	0.332	6.206	3.852
36	4.524	4.132	2.994	3.64	4.766	4.358	2.862	5.873	3.553
37	8.192	8.841	4.511	7.308	8.481	7.071	7.423	6.595	5.048
38	3.778	4.686	1.771	2.95	4.067	2.05	3.391	4.195	1.758
39	3.332	4.115	1.92	2.465	3.621	1.413	3.033	4.426	1.989
40	3.882	3.484	2.949	3.191	4.187	3.91	2.135	5.872	3.507
41	4.537	4.957	1.003	3.653	4.826	3.433	3.538	3.287	1.393
42	12.73	13.496	9.601	11.846	13.015	10.954	12.317	6.422	8.859
43	5.407	6.173	2.314	4.523	5.692	3.925	5.088	4.152	1.799
44	3.427	3.812	1.722	2.543	3.687	3.411	2.393	4.432	2.112
45	2.348	3.114	1.732	1.464	2.637	1.8	2.029	4.655	2.218

Table B.1 continued from previous page

46	2.651	3.417	1.845	1.767	2.94	1.431	2.397	4.768	2.331
47	4.263	5.111	4.715	4.107	4.479	2.264	5.134	7.245	4.808
48	4.519	5.285	1.818	3.635	4.808	2.708	4.175	3.747	1.394
49	3.265	2.789	3.862	2.627	3.56	4.47	1.539	6.382	4.054
50	3.543	3.066	3.344	3.001	3.848	4.38	1.796	6.267	3.939
51	6.536	7.384	6.157	6.021	6.752	4.178	7.407	8.725	6.288
52	2.007	1.531	3.404	1.504	2.302	3.305	0.27	6.327	3.96
53	2.764	3.4	1.45	1.88	3.053	2.442	2.13	4.373	2.044
54	4.823	5.589	1.743	3.939	5.108	3.354	4.504	3.581	1.228
55	2.301	3.221	3.603	1.479	2.59	1.412	2.585	6.469	4.032
56	2.671	3.437	2.411	1.787	2.96	2.128	2.675	5.334	2.897
57	2.918	3.684	1.877	2.034	3.208	1.844	2.602	4.704	2.267
58	5.013	5.654	1.331	4.129	5.302	4.15	4.243	3.607	1.868
59	1.826	2.746	3.436	1.004	2.115	0.937	2.643	5.994	3.557
60	8.146	7.934	6.713	7.262	8.388	7.98	6.484	9.592	7.201
61	2.633	3.309	2.017	1.749	2.922	2.085	2.039	4.816	2.379
62	2.049	2.713	2.165	1.165	2.339	2.413	1.443	5.088	2.759
63	3.162	3.169	2.036	2.278	3.451	2.716	2.049	4.96	2.523
64	2.707	3.306	2.242	1.823	2.996	2.595	2.056	5.069	2.705
65	10.935	11.701	7.822	10.051	11.22	9.175	10.538	4.643	7.08
66	3.576	4.096	1.281	2.692	3.865	3.475	2.826	4.038	1.671
67	8.399	9.165	4.868	7.515	8.684	6.638	8.001	1.679	4.116
68	8.348	9.087	4.538	7.464	8.637	7.188	7.817	1.349	3.786
69	0.508	1.196	3.879	1.179	0.796	2.468	1.776	6.802	4.365
70	4.105	4.073	2.651	3.221	4.394	3.987	2.803	5.483	3.046
71	4.23	4.068	2.431	3.346	4.503	4.096	2.798	5.288	2.917
72	4.24	3.848	2.781	3.356	4.482	4.074	2.578	5.638	3.267
73	4.522	5.442	4.948	3.629	4.811	2.586	5.119	7.372	4.935
74	6.22	7.14	6.144	5.404	6.509	4.165	7.038	8.712	6.275

Table B.1 continued from previous page

75	2.691	2.721	3.175	2.194	2.981	3.423	1.471	6.098	3.711
76	2.56	2.193	2.657	1.676	2.849	2.905	0.923	5.58	3.157
77	2.353	3.273	2.797	1.958	2.643	3.229	2.259	5.221	2.784
78	2.355	3.121	2.061	1.471	2.645	2.34	2.393	4.918	2.481
79	2.241	3.161	3.295	1.425	2.53	0.516	3.059	5.719	3.282
80	2.48	3.4	2.733	1.855	2.77	3.103	2.133	5.348	2.911
81	5.179	4.835	3.87	4.295	5.469	5.061	3.496	6.749	4.358
82	4.162	4.842	0.521	3.278	4.451	3.058	3.572	2.945	0.508
83	4.946	5.866	5.471	4.13	5.235	3.01	5.764	8.001	5.564
84	0.767	1.687	3.845	1.224	1.056	2.434	1.979	6.768	4.331
85	0.939	1.859	3.294	0.544	1.229	2.196	1.654	6.217	3.78
86	4.369	4.337	2.915	3.485	4.658	4.251	3.067	5.834	3.397
87	1.859	2.625	2.644	0.975	2.148	2.749	1.863	5.567	3.13
88	4.998	5.846	4.922	3.97	5.214	2.544	5.524	7.49	5.077
89	3.83	4.282	3.743	3.158	4.358	1.315	4.295	6.289	3.852
90	2.75	2.819	2.782	1.866	3.039	3.114	1.569	5.705	3.371
91	2.531	3.195	2.525	1.647	2.82	2.895	1.925	5.448	3.011
92	3.226	3.755	1.729	2.342	3.516	3.21	2.505	4.652	2.216
93	4.43	5.143	0.748	3.546	4.719	3.308	3.842	3.541	1.234
94	3.161	2.791	4.365	2.629	3.466	4.472	1.299	7.288	4.852
95	3.834	3.756	4.628	3.501	4.308	5.344	2.555	7.253	5.01
96	4.385	3.923	4.044	3.825	4.69	5.145	2.638	6.923	4.532
97	5.109	5.875	1.299	4.225	5.398	3.949	4.579	2.146	0.547
98	5.183	6.031	5.457	4.486	5.399	3.029	6.009	8.003	5.566
99	4.457	4.059	3.386	3.627	4.753	4.345	2.71	6.309	3.873
100	4.957	4.559	3.584	4.127	5.253	4.845	3.21	6.463	4.077
101	3.335	2.858	4.722	2.793	3.64	4.636	1.284	7.645	5.278
102	2.499	2.022	3.886	1.957	2.804	3.8	0.448	6.809	4.442
103	2.777	2.332	4.196	2.235	3.114	4.078	0.726	7.119	4.72

Table B.1 continued from previous page

104	3.629	3.152	5.016	3.087	3.934	4.93	1.578	7.939	5.572
105	4.004	4.381	1.719	3.12	4.264	3.988	2.97	4.003	2.109
106	2.765	3.401	1.842	1.881	3.055	2.749	2.131	4.599	2.162
107	3.67	4.047	1.858	2.786	3.93	3.523	2.636	4.715	2.344
108	4.021	4.29	2.209	3.137	4.281	3.874	2.983	5.066	2.695
109	3.762	4.139	1.95	2.878	4.022	3.615	2.728	4.807	2.436
110	3.609	4.061	3.632	3.047	4.137	1.204	4.122	6.178	3.741
111	6.999	7.765	3.189	6.115	7.288	5.839	6.468	0	2.437
112	0.29	0.742	4.255	1.604	0.818	2.936	1.847	7.178	4.741
113	5.125	4.913	3.691	4.241	5.367	4.959	3.463	6.57	4.179
114	9.265	9.053	7.832	8.381	9.507	9.099	7.603	10.711	8.32
115	5.337	5.839	1.527	4.453	5.626	4.177	4.569	2.45	0.775
116	1.248	0.217	5.011	2.561	1.776	3.803	1.877	7.934	5.519
117	1.74	1.264	3.867	1.382	2.035	3.225	0.795	6.79	4.353
118	5.149	5.685	1.517	4.265	5.438	3.338	4.508	3.117	0.764
119	5.337	5.757	1.662	4.453	5.626	4.223	4.338	4.087	2.148
120	8.109	8.757	4.427	7.225	8.398	6.987	7.34	3.97	5.021
121	12.423	13.072	8.742	11.539	12.712	11.302	11.654	10.826	9.228
122	5.621	6.387	2.325	4.737	5.906	3.947	5.302	4.176	1.823
123	2.9	3.666	1.176	2.016	3.189	2.1	2.584	4.099	1.662
124	2.683	2.23	2.734	1.847	2.973	3.076	0.98	5.657	3.22
125	5.964	6.432	2.089	5.08	6.253	4.649	5.013	2.685	2.575
126	4.088	4.917	2.634	3.174	4.377	1.331	4.906	5.226	2.789
127	1.846	2.612	2.52	0.962	2.135	2.736	1.85	5.443	3.006
128	1.342	1.728	3.309	0.825	1.632	2.668	0.979	6.232	3.795
129	4.486	4.246	3.129	3.602	4.728	4.32	2.824	6.008	3.617
NODES	27	28	29	30	31	32	33	34	35
depot	4.35	3.986	4.132	4.058	8.829	4.566	7.536	7.387	5.414

Table B.1 continued from previous page

1	0.577	0.888	0.786	0.68	4.618	1.374	3.918	3.769	1.859
2	1.585	1.949	2.006	1.9	4.38	2.059	4.109	4.162	1.436
3	1.901	1.861	2.598	2.492	4.919	1.336	4.999	5.052	0.546
4	3.174	3.538	3.595	3.489	4.969	3.116	5.698	5.751	2.298
5	0.889	0.848	1.586	1.48	5.626	0.754	4.851	4.688	1.682
6	7.637	7.931	7.553	7.447	8.383	8.434	4.746	4.17	8.753
7	4.952	4.588	4.657	4.763	9.689	5.411	8.591	8.442	5.983
8	8.426	8.72	8.342	8.236	9.172	9.223	5.535	4.959	9.448
9	2.313	2.677	2.752	2.646	4.505	2.3	4.88	4.933	1.482
10	2.616	2.98	3.084	2.978	4.337	2.501	5.183	5.236	1.683
11	2.719	3.083	3.14	3.034	4.102	2.785	5.402	5.277	1.995
12	0.449	0.813	1.146	1.04	4.744	1.316	4.225	4.165	1.741
13	3.289	3.583	2.805	2.699	6.078	4.086	4.153	3.99	4.672
14	3.244	3.538	2.759	2.653	5.997	4.041	4.082	3.933	4.627
15	2.02	2.336	2.583	2.477	3.365	2.839	2.312	2.25	3.176
16	2.407	2.701	2.096	1.99	5.171	3.274	2.724	2.585	3.79
17	3.305	3.669	3.726	3.62	5.1	3.247	5.825	5.878	2.429
18	2.955	3.319	3.376	3.27	4.75	2.897	5.479	5.532	2.079
19	3.064	3.107	3.761	3.655	5.55	2.499	6.245	6.298	1.709
20	1.998	2.194	1.816	1.71	4.323	2.697	2.399	2.236	3.283
21	2.071	2.435	2.492	2.386	4.187	2.337	4.595	4.648	1.547
22	3.197	3.561	3.636	3.53	4.918	3.202	5.764	5.817	2.384
23	2.79	3.153	3.36	3.254	2.344	3.657	4.01	3.939	3.39
24	1.871	1.657	2.342	2.236	5.557	1.15	5.16	5.213	0.332
25	4.823	5.117	4.695	4.589	6.891	5.62	4.237	4.098	6.206
26	2.388	2.751	2.309	2.203	4.454	3.254	1.884	1.745	3.852
27	0	0.364	0.697	0.591	4.738	0.867	4.018	3.869	1.756
28	0.364	0	0.991	0.885	5.101	1.231	4.312	4.163	1.989
29	0.697	0.991	0	0.106	5.295	1.564	3.934	3.785	2.394

Table B.1 continued from previous page

30	0.591	0.885	0.106	0	5.189	1.458	3.828	3.679	2.288
31	4.738	5.101	5.295	5.189	0	5.604	4.922	4.851	5.465
32	0.867	1.231	1.564	1.458	5.604	0	4.815	4.666	1.191
33	4.018	4.312	3.934	3.828	4.922	4.815	0	0.576	4.932
34	3.869	4.163	3.785	3.679	4.851	4.666	0.576	0	4.985
35	1.756	1.989	2.394	2.288	5.465	1.191	4.932	4.985	0
36	1.569	1.205	1.704	1.598	6.306	2.029	5.112	4.963	2.6
37	5.778	6.072	5.294	5.188	8.567	6.575	6.628	6.479	7.161
38	2.172	2.488	2.735	2.629	3.245	2.991	2.202	2.131	3.163
39	1.557	1.851	2.136	2.03	3.344	2.354	2.597	2.526	2.805
40	1.12	0.913	1.817	1.711	5.857	0.985	5.067	4.918	1.913
41	1.893	2.187	1.408	1.302	4.911	2.69	2.972	2.823	3.276
42	11.067	11.361	10.983	10.877	11.813	11.864	8.176	7.6	12.089
43	3.933	4.227	3.849	3.743	4.837	4.73	1.24	1.281	4.86
44	0.748	1.042	0.263	0.157	5.346	1.545	3.692	3.543	2.131
45	1.089	1.453	1.56	1.454	3.735	1.956	3.131	3.184	1.801
46	1.458	1.822	1.929	1.823	3.366	2.325	3.227	3.156	2.169
47	4.3	4.663	4.776	4.67	2.345	5.166	5.416	5.336	4.915
48	2.852	3.146	3.367	3.261	3.62	3.649	1.38	1.231	3.843
49	1.947	1.906	2.644	2.538	6.517	1.241	5.577	5.428	1.432
50	1.59	1.473	2.287	2.181	6.315	0.723	5.462	5.313	1.534
51	6.572	6.935	7.072	6.966	1.977	7.438	6.756	6.66	7.188
52	1.601	1.927	2.298	2.192	5.287	1.036	5.032	5.085	0.246
53	1.116	1.41	1.412	1.306	4.377	1.913	3.567	3.418	1.846
54	3.362	3.656	3.278	3.172	4.266	4.159	0.656	0.709	4.276
55	2.482	2.846	2.947	2.841	3.756	2.956	4.587	4.525	2.605
56	1.772	2.136	2.243	2.137	4.029	2.639	3.814	3.867	2.484
57	1.663	2.027	2.102	1.996	3.463	2.53	3.028	2.957	2.374
58	2.598	2.892	2.125	2.019	5.387	3.395	3.462	3.299	3.981

Table B.1 continued from previous page

59	2.957	3.321	3.422	3.316	3.281	3.34	4.112	4.05	2.55
60	5.191	4.827	4.897	5.003	9.928	5.651	8.831	8.682	6.672
61	0.804	1.168	1.275	1.169	4.02	1.671	3.416	3.469	1.811
62	1.162	1.456	1.327	1.221	4.348	1.38	3.745	3.798	1.215
63	0.254	0.548	0.833	0.727	4.647	1.051	3.9	3.829	2.01
64	0.318	0.682	1.015	0.909	4.543	1.185	4.024	3.964	1.942
65	9.288	9.582	9.204	9.098	10.033	10.085	6.397	5.821	10.31
66	1.181	1.475	1.286	1.18	5.215	1.978	3.314	3.151	2.564
67	6.502	6.796	6.374	6.268	7.55	7.299	3.861	3.285	7.773
68	6.172	6.466	6.044	5.938	8.24	6.969	4.583	4.007	7.555
69	2.711	3.075	3.179	3.073	4.432	2.474	5.278	5.331	1.684
70	1.456	1.146	1.423	1.317	5.922	1.97	4.583	4.42	2.541
71	1.505	1.141	1.079	0.973	6.031	1.965	4.548	4.399	2.536
72	1.285	0.921	1.429	1.323	6.009	1.745	4.898	4.749	2.316
73	4.204	4.568	4.808	4.702	2.856	5.071	5.748	5.677	4.891
74	6.33	6.693	6.817	6.711	3.669	7.196	6.733	6.647	6.9
75	1.815	1.989	2.468	2.362	5.358	1.528	4.755	4.808	1.357
76	1.493	1.665	1.819	1.713	4.84	1.118	4.237	4.29	0.695
77	1.449	1.813	2.067	1.961	5.164	1.459	4.321	4.158	2.031
78	1.012	1.306	1.177	1.071	4.275	1.809	3.769	3.822	2.165
79	2.577	2.941	3.274	3.168	2.86	3.444	4.526	4.455	2.966
80	1.323	1.687	1.941	1.835	5.038	1.333	4.278	4.285	1.905
81	2.272	1.908	2.054	2.16	6.996	2.346	5.988	5.839	3.275
82	1.987	2.281	1.903	1.797	3.97	2.784	2.045	1.882	3.344
83	5.056	5.419	5.555	5.449	3.101	5.922	6.165	6.079	5.671
84	2.724	3.088	3.145	3.039	4.579	2.721	5.248	5.301	1.921
85	2.173	2.537	2.594	2.488	4.54	2.352	4.697	4.75	1.562
86	1.72	1.41	1.687	1.581	6.186	2.234	4.943	4.78	2.805
87	1.233	1.597	1.852	1.746	4.673	1.96	4.142	4.195	1.759

Table B.1 continued from previous page

88	4.486	4.849	5.056	4.95	2.448	5.352	5.521	5.45	5.296
89	3.257	3.62	3.827	3.721	1.481	4.123	4.46	4.389	4.067
90	1.31	0.996	1.681	1.575	5.049	0.679	4.446	4.499	1.364
91	1.115	1.479	1.733	1.627	4.83	1.541	4.07	4.123	1.697
92	0.692	1.055	0.505	0.399	5.145	1.558	3.847	3.698	2.33
93	2.16	2.491	1.712	1.606	4.78	2.994	2.865	2.716	3.58
94	2.791	2.477	3.162	3.056	6.436	1.97	6.014	6.067	1.082
95	2.958	2.74	3.425	3.319	7.308	2.091	6.547	6.384	2.497
96	2.355	2.082	2.228	2.334	7.092	1.488	6.162	6.013	2.391
97	2.935	3.298	2.805	2.699	5.001	3.801	2.347	2.208	4.399
98	4.971	5.334	5.541	5.435	3.109	5.837	6.174	6.103	5.781
99	1.556	1.192	1.772	1.878	6.293	1.56	5.504	5.355	2.489
100	2.056	1.692	1.768	1.874	6.793	2.06	5.702	5.553	2.989
101	3.012	2.941	3.626	3.52	6.802	2.434	6.358	6.411	1.616
102	2.176	2.105	2.79	2.684	5.973	1.598	5.522	5.575	0.78
103	2.454	2.383	3.068	2.962	6.197	1.876	5.8	5.853	1.058
104	3.306	3.235	3.92	3.814	7.049	2.728	6.652	6.705	1.91
105	1.325	1.619	0.84	0.734	5.627	2.122	3.688	3.539	2.708
106	0.486	0.78	0.611	0.505	4.684	1.283	3.699	3.536	1.869
107	0.991	1.285	0.506	0.4	5.458	1.788	3.975	3.826	2.374
108	1.342	1.326	0.857	0.751	5.809	2.139	4.326	4.177	2.725
109	1.083	1.377	0.598	0.492	5.55	1.88	4.067	3.918	2.466
110	3.146	3.509	3.716	3.61	2.124	4.012	4.349	4.278	3.956
111	4.823	5.117	4.695	4.589	6.891	5.62	4.237	4.098	6.206
112	2.908	3.114	3.555	3.449	4.808	2.607	5.654	5.707	1.789
113	2.17	1.806	1.875	1.981	6.907	2.629	5.809	5.66	3.65
114	6.31	5.946	6.016	6.122	11.047	6.77	9.95	9.801	7.791
115	2.924	3.218	2.612	2.506	5.229	3.791	2.576	2.437	4.307
116	3.281	3.144	3.829	3.723	5.765	2.637	6.462	6.515	1.819

Table B.1 continued from previous page

117	2.282	2.062	2.747	2.641	5.189	1.555	5.266	5.319	0.737
118	3.152	3.515	3.073	2.967	4.147	4.018	1.12	0.981	4.394
119	2.693	2.987	2.208	2.102	5.694	3.49	3.772	3.623	4.076
120	5.695	5.989	5.21	5.104	8.459	6.492	6.544	6.395	7.078
121	10.009	10.303	9.525	9.419	12.774	10.806	10.859	10.71	11.392
122	3.958	4.252	3.874	3.768	4.703	4.755	1.067	0.491	5.074
123	1.559	1.922	2.084	1.978	4.035	2.425	3.293	3.144	2.356
124	1.324	1.498	1.99	1.884	5.011	1.181	4.279	4.332	0.866
125	3.368	3.662	2.883	2.777	6.121	4.165	4.206	4.057	4.751
126	3.585	3.934	4.184	4.078	1.689	4.437	3.233	3.162	4.719
127	1.109	1.473	1.728	1.622	4.614	1.976	4.129	4.182	1.635
128	1.962	2.246	2.609	2.503	4.632	1.739	4.708	4.761	0.921
129	1.531	1.167	1.313	1.419	6.268	1.747	5.247	5.098	2.676
NODES	36	37	38	39	40	41	42	43	44
depot	3.832	6.616	6.269	5.837	3.582	5.086	14.585	7.451	3.941
1	1.964	5.832	2.165	1.878	1.626	1.982	10.967	3.846	0.837
2	3.154	6.822	2.55	1.979	2.705	3.167	11.36	4.037	2.057
3	3.066	7.679	3.44	2.869	2.321	3.794	12.25	4.927	2.649
4	4.743	8.411	3.997	3.551	4.101	4.756	12.949	5.626	3.646
5	1.593	6.667	3.013	2.376	0.548	2.782	11.886	4.766	1.637
6	8.731	9.587	5.687	6.214	8.686	6.28	3.43	5.049	7.311
7	4.4	6.002	7.076	6.439	4.742	6.065	15.64	8.506	4.92
8	9.52	10.375	6.476	6.9	9.475	7.068	3.679	5.821	8.1
9	3.882	7.597	3.176	2.736	3.285	3.942	12.131	4.795	2.803
10	4.185	7.9	3.48	3.04	3.486	4.245	12.434	5.098	3.106
11	4.288	7.956	3.245	2.805	3.77	4.301	12.292	5.317	3.191
12	2.018	6.227	2.178	2.006	1.569	2.342	11.151	4.153	1.197
13	3.983	2.637	3.358	3.548	4.338	1.408	10.72	4.068	2.542

Table B.1 continued from previous page

14	3.937	4.458	3.032	3.112	4.293	1.351	8.751	3.997	2.496
15	3.541	5.752	0.152	0.485	3.139	2.097	9.237	2.227	2.49
16	3.274	4.343	2.475	2.287	3.456	0.688	9.021	2.639	1.833
17	4.874	8.542	4.122	3.682	4.232	4.887	13.076	5.753	3.777
18	4.524	8.192	3.778	3.332	3.882	4.537	12.73	5.407	3.427
19	4.132	8.841	4.686	4.115	3.484	4.957	13.496	6.173	3.812
20	2.994	4.511	1.771	1.92	2.949	1.003	9.601	2.314	1.722
21	3.64	7.308	2.95	2.465	3.191	3.653	11.846	4.523	2.543
22	4.766	8.481	4.067	3.621	4.187	4.826	13.015	5.692	3.687
23	4.358	7.071	2.05	1.413	3.91	3.433	10.954	3.925	3.411
24	2.862	7.423	3.391	3.033	2.135	3.538	12.317	5.088	2.393
25	5.873	6.595	4.195	4.426	5.872	3.287	6.422	4.152	4.432
26	3.553	5.048	1.758	1.989	3.507	1.393	8.859	1.799	2.112
27	1.569	5.778	2.172	1.557	1.12	1.893	11.067	3.933	0.748
28	1.205	6.072	2.488	1.851	0.913	2.187	11.361	4.227	1.042
29	1.704	5.294	2.735	2.136	1.817	1.408	10.983	3.849	0.263
30	1.598	5.188	2.629	2.03	1.711	1.302	10.877	3.743	0.157
31	6.306	8.567	3.245	3.344	5.857	4.911	11.813	4.837	5.346
32	2.029	6.575	2.991	2.354	0.985	2.69	11.864	4.73	1.545
33	5.112	6.628	2.202	2.597	5.067	2.972	8.176	1.24	3.692
34	4.963	6.479	2.131	2.526	4.918	2.823	7.6	1.281	3.543
35	2.6	7.161	3.163	2.805	1.913	3.276	12.089	4.86	2.131
36	0	6.202	3.693	3.056	1.553	2.586	12.161	5.027	1.441
37	6.202	0	5.616	5.658	6.828	3.886	13.016	6.543	5.031
38	3.693	5.616	0	0.637	3.291	2.191	9.117	2.117	2.642
39	3.056	5.658	0.637	0	2.677	2.381	9.541	2.512	2.187
40	1.553	6.828	3.291	2.677	0	2.942	12.116	4.982	1.797
41	2.586	3.886	2.191	2.381	2.942	0	9.709	2.887	1.145
42	12.161	13.016	9.117	9.541	12.116	9.709	0	8.479	10.741

Table B.1 continued from previous page

43	5.027	6.543	2.117	2.512	4.982	2.887	8.479	0	3.62
44	1.441	5.031	2.642	2.187	1.797	1.145	10.741	3.62	0
45	2.658	5.848	1.707	1.07	2.209	2.193	10.382	3.059	1.611
46	2.992	5.961	1.338	0.701	2.578	2.306	10.242	3.155	1.98
47	5.862	8.477	3.346	2.819	5.419	5.176	12.309	5.324	4.827
48	4.351	6.061	0.9	1.295	3.972	2.405	8.429	1.308	3.125
49	2.399	7.485	4.071	3.434	1.606	3.6	12.626	5.505	2.695
50	2.218	7.298	3.638	3.001	1.173	3.413	12.511	5.39	2.268
51	8.135	10.377	5.079	5.092	7.691	6.721	13.646	6.661	7.123
52	2.807	7.379	3.368	2.913	2.021	3.494	12.283	4.96	2.349
53	2.379	5.715	1.539	1.671	2.165	2.209	10.512	3.495	1.463
54	4.456	5.972	1.546	1.941	4.411	2.316	7.907	0.584	3.036
55	4.051	7.719	2.427	2.55	3.602	4.064	11.474	4.502	2.998
56	3.341	6.527	2.002	1.365	2.892	2.872	10.906	3.742	2.294
57	3.008	5.936	1.002	0.431	2.783	2.281	9.972	2.943	2.153
58	3.303	3.328	2.667	2.857	3.647	0.717	10.029	3.377	1.862
59	4.526	7.552	1.952	2.285	4.077	3.897	11.037	4.027	3.473
60	4.64	7.423	7.315	6.678	4.812	6.305	15.88	8.746	5.16
61	2.373	6.133	1.861	1.29	1.924	2.471	10.667	3.344	1.326
62	2.505	6.375	2.141	1.618	2.214	2.523	10.996	3.673	1.378
63	1.753	5.915	1.94	1.303	1.374	2.029	10.844	3.815	0.884
64	1.887	6.096	1.977	1.875	1.438	2.211	10.95	3.952	1.066
65	10.382	11.237	7.338	7.762	10.337	7.93	3.967	6.683	8.962
66	2.464	4.81	2.433	2.565	2.23	1.155	10.349	3.229	1.206
67	7.552	8.273	4.83	5.225	7.551	4.966	5.319	4.147	6.111
68	7.222	7.943	5.544	5.775	7.221	4.636	5.073	4.869	5.781
69	4.173	7.995	3.575	3.135	3.459	4.34	12.529	5.193	3.201
70	0.861	5.995	3.482	2.895	1.494	2.444	11.618	4.498	1.299
71	0.797	5.552	3.411	2.992	1.489	2.001	11.597	4.476	0.856

Table B.1 continued from previous page

72	0.349	5.902	3.409	2.772	1.269	2.351	11.947	4.826	1.206
73	5.773	8.809	3.678	3.151	5.324	5.368	12.692	5.663	4.859
74	7.892	10.364	5.066	4.849	7.449	6.708	13.633	6.648	6.868
75	3.194	7.341	3.092	2.628	2.362	3.664	12.006	4.683	2.519
76	2.825	6.867	2.633	2.11	1.952	3.015	11.488	4.165	1.87
77	3.018	6.873	2.726	2.434	2.293	3.217	11.356	3.65	2.118
78	2.355	6.134	1.927	1.356	2.061	2.373	10.897	3.697	1.228
79	4.146	7.371	2.373	1.929	3.697	3.715	11.458	4.441	3.325
80	2.892	6.849	2.6	2.308	2.167	3.137	11.483	4.192	1.992
81	1.797	7.285	4.396	3.759	1.362	3.462	13.037	5.903	2.317
82	3.081	4.597	1.25	1.645	3.036	0.941	9.08	1.96	1.661
83	6.618	9.233	4.102	3.575	6.175	5.932	13.065	6.08	5.606
84	4.293	7.961	3.547	3.118	3.706	4.306	12.499	5.176	3.196
85	3.742	7.41	3.138	2.567	3.293	3.755	11.948	4.625	2.645
86	1.125	6.259	3.746	3.159	1.758	2.708	11.978	4.858	1.563
87	2.802	6.76	2.442	2.002	2.353	3.048	11.393	4.057	1.903
88	6.054	8.75	3.729	3.092	5.605	5.433	12.465	5.436	5.107
89	4.825	7.521	2.303	1.863	4.376	4.204	11.388	4.375	3.878
90	2.201	6.762	2.611	2.319	1.514	2.877	11.575	4.374	1.732
91	2.684	6.641	2.392	2.1	2.235	2.929	11.321	3.984	1.784
92	1.465	5.587	2.585	2.249	1.812	1.701	10.896	3.762	0.556
93	2.796	4.313	1.815	1.895	3.246	0.658	9.914	2.78	1.449
94	3.682	8.243	4.207	3.887	2.955	4.358	13.171	5.942	3.213
95	3.625	8.507	4.913	4.285	2.542	4.621	13.582	6.462	3.476
96	1.971	7.459	4.479	3.842	1.477	3.636	13.211	6.077	2.491
97	3.983	5.052	2.305	2.536	4.054	1.397	8.568	2.262	2.542
98	6.539	9.235	4.017	3.577	6.09	5.918	13.102	6.089	5.592
99	1.515	7.003	3.68	3.043	0.576	3.18	12.553	5.419	2.035
100	1.511	6.999	4.18	3.543	1.076	3.176	12.751	5.617	2.031

Table B.1 continued from previous page

101	4.125	8.707	4.598	4.231	3.419	4.822	13.547	6.286	3.677
102	3.289	7.871	3.839	3.395	2.583	3.986	12.679	5.45	2.841
103	3.567	8.149	3.993	3.673	2.861	4.264	12.957	5.728	3.119
104	4.419	9.001	4.847	4.525	3.713	5.116	13.811	6.58	3.971
105	2.018	4.602	2.646	2.764	2.374	0.716	10.425	3.603	0.577
106	1.789	5.659	2.124	1.922	1.535	1.807	10.734	3.614	0.662
107	1.223	5.163	2.925	2.43	2.062	1.428	11.024	3.903	0.283
108	0.872	5.33	3.189	2.781	1.711	1.779	11.375	4.254	0.634
109	1.131	5.071	2.93	2.522	1.97	1.52	11.116	3.995	0.375
110	4.714	7.41	2.192	1.752	4.265	4.093	11.277	4.264	3.767
111	5.873	6.595	4.195	4.426	5.872	3.287	6.422	4.152	4.432
112	4.319	8.371	3.951	3.511	3.592	4.716	12.905	5.569	3.606
113	1.618	5.487	4.294	3.657	1.959	3.283	12.858	5.724	2.138
114	5.759	8.542	8.434	7.797	5.931	7.424	16.999	9.865	6.279
115	3.79	4.859	2.533	2.764	3.973	1.204	8.872	2.491	2.349
116	4.349	8.91	4.903	4.332	3.622	5.025	13.713	6.39	3.88
117	3.267	7.828	3.566	3.126	2.54	3.943	12.517	5.181	2.798
118	4.317	5.812	1.451	1.925	4.271	2.157	8.179	1.035	2.876
119	3.386	3.356	2.73	2.81	3.742	0.8	10.509	3.687	1.945
120	6.388	2.634	5.494	5.574	6.744	3.802	10.391	6.459	4.947
121	10.433	4.231	9.809	9.889	11.059	8.117	17.247	10.774	9.262
122	5.052	6.568	2.007	2.534	5.007	2.912	7.284	1.37	3.632
123	3.127	5.292	1.239	1.163	2.678	1.637	10.203	3.208	2.135
124	2.703	6.85	2.601	2.281	2.016	3.186	11.53	4.194	2.041
125	3.798	4.582	3.156	3.236	4.417	1.475	9.107	4.121	2.62
126	5.139	6.878	1.556	2.083	4.704	3.222	10.177	3.148	3.942
127	2.678	6.636	2.429	1.989	2.229	2.924	11.38	4.044	1.779
128	3.451	7.425	3.008	2.568	2.724	3.77	11.959	4.623	2.66
129	1.056	6.544	3.655	3.018	0.763	2.721	12.296	5.162	1.576

Table B.1 continued from previous page

NODES	45	46	47	48	49	50	51	52	53
depot	5.094	5.463	8.31	6.969	5.109	4.591	10.606	5.556	4.854
1	0.884	1.252	4.121	2.89	2.399	2.097	6.395	1.967	0.626
2	0.978	1.281	3.941	3.149	2.406	2.78	6.357	1.258	1.394
3	1.868	2.171	4.651	4.039	1.626	1.982	6.896	0.368	2.284
4	2.567	2.87	4.409	4.738	3.484	3.762	6.682	2.226	2.983
5	1.976	2.345	5.188	3.671	1.058	0.943	7.46	1.79	1.935
6	6.952	6.915	8.88	4.999	9.196	9.081	10.217	8.853	7.083
7	6.039	6.408	9.251	7.734	6.033	5.6	11.523	6.229	5.697
8	7.725	7.601	9.669	5.788	9.985	9.87	11.005	9.556	7.872
9	1.753	2.056	3.594	3.924	2.683	2.946	5.867	1.425	2.132
10	2.056	2.359	3.898	4.227	2.859	3.147	6.171	1.601	2.435
11	2.512	2.815	3.663	4.046	3.075	3.431	6.079	1.749	2.717
12	1.094	1.463	4.306	2.934	2.396	2.039	6.578	1.642	1.335
13	3.36	3.473	6.343	3.572	4.848	4.733	7.912	4.814	3.078
14	3.362	3.475	5.931	3.515	4.951	4.764	7.831	4.845	3.318
15	1.555	1.186	3.194	1.019	3.919	3.486	5.199	3.216	1.387
16	2.477	2.59	5.106	2.234	4.179	3.997	7.005	4.008	2.493
17	2.698	3.001	4.582	4.869	3.615	3.893	6.855	2.357	3.114
18	2.348	2.651	4.263	4.519	3.265	3.543	6.536	2.007	2.764
19	3.114	3.417	5.111	5.285	2.789	3.066	7.384	1.531	3.4
20	1.732	1.845	4.715	1.818	3.862	3.344	6.157	3.404	1.45
21	1.464	1.767	4.107	3.635	2.627	3.001	6.021	1.504	1.88
22	2.637	2.94	4.479	4.808	3.56	3.848	6.752	2.302	3.053
23	1.8	1.431	2.264	2.708	4.47	4.38	4.178	3.305	2.442
24	2.029	2.397	5.134	4.175	1.539	1.796	7.407	0.27	2.13
25	4.655	4.768	7.245	3.747	6.382	6.267	8.725	6.327	4.373
26	2.218	2.331	4.808	1.394	4.054	3.939	6.288	3.96	2.044

Table B.1 continued from previous page

27	1.089	1.458	4.3	2.852	1.947	1.59	6.572	1.601	1.116
28	1.453	1.822	4.663	3.146	1.906	1.473	6.935	1.927	1.41
29	1.56	1.929	4.776	3.367	2.644	2.287	7.072	2.298	1.412
30	1.454	1.823	4.67	3.261	2.538	2.181	6.966	2.192	1.306
31	3.735	3.366	2.345	3.62	6.517	6.315	1.977	5.287	4.377
32	1.956	2.325	5.166	3.649	1.241	0.723	7.438	1.036	1.913
33	3.131	3.227	5.416	1.38	5.577	5.462	6.756	5.032	3.567
34	3.184	3.156	5.336	1.231	5.428	5.313	6.66	5.085	3.418
35	1.801	2.169	4.915	3.843	1.432	1.534	7.188	0.246	1.846
36	2.658	2.992	5.862	4.351	2.399	2.218	8.135	2.807	2.379
37	5.848	5.961	8.477	6.061	7.485	7.298	10.377	7.379	5.715
38	1.707	1.338	3.346	0.9	4.071	3.638	5.079	3.368	1.539
39	1.07	0.701	2.819	1.295	3.434	3.001	5.092	2.913	1.671
40	2.209	2.578	5.419	3.972	1.606	1.173	7.691	2.021	2.165
41	2.193	2.306	5.176	2.405	3.6	3.413	6.721	3.494	2.209
42	10.382	10.242	12.309	8.429	12.626	12.511	13.646	12.283	10.512
43	3.059	3.155	5.324	1.308	5.505	5.39	6.661	4.96	3.495
44	1.611	1.98	4.827	3.125	2.695	2.268	7.123	2.349	1.463
45	0	0.369	3.239	2.171	2.783	2.679	5.512	1.901	0.642
46	0.369	0	2.87	1.925	3.151	2.949	5.143	2.269	1.011
47	3.239	2.87	0	4.114	5.998	5.796	4.264	5.019	3.858
48	2.171	1.925	4.114	0	4.729	4.296	5.429	4.072	2.295
49	2.783	3.151	5.998	4.729	0	0.518	8.268	1.326	2.675
50	2.679	2.949	5.796	4.296	0.518	0	8.092	1.682	2.56
51	5.512	5.143	4.264	5.429	8.268	8.092	0	7.264	6.154
52	1.901	2.269	5.019	4.072	1.326	1.682	7.264	0	1.954
53	0.642	1.011	3.858	2.295	2.675	2.56	6.154	1.954	0
54	2.475	2.571	4.76	0.724	4.921	4.806	6.1	4.376	2.911
55	1.875	2.178	3.676	3.294	3.616	3.679	5.59	2.359	2.291

Table B.1 continued from previous page

56	0.683	0.766	3.51	2.66	3.466	3.362	5.863	2.449	1.325
57	1.144	0.816	2.944	1.726	3.356	3.253	5.297	2.482	1.522
58	2.669	2.782	5.652	2.881	4.157	4.042	7.221	4.123	2.387
59	2.35	2.368	3.201	2.819	3.63	3.986	5.115	2.373	2.766
60	6.28	6.649	9.49	7.973	6.273	5.84	11.762	6.555	6.237
61	0.285	0.654	3.501	2.456	2.669	2.394	5.797	1.919	0.663
62	0.614	0.982	3.829	2.785	2.197	2.103	6.125	1.323	0.715
63	1.342	1.613	4.122	2.598	2.131	1.698	6.395	1.779	1
64	0.893	1.262	4.105	2.733	2.265	1.908	6.377	1.843	1.134
65	8.587	8.392	10.531	6.65	10.847	10.732	11.868	10.418	8.734
66	1.675	2.044	4.892	2.733	2.74	2.625	7.073	2.706	1.057
67	6.051	5.855	8.044	4.124	8.061	7.946	9.384	7.881	6.052
68	6.004	6.117	8.594	4.846	7.731	7.616	10.074	7.676	5.722
69	2.151	2.454	3.993	4.322	2.764	3.12	6.266	1.506	2.53
70	2.285	2.556	5.426	4.002	2.275	2.159	7.699	2.683	1.943
71	2.394	2.665	5.535	3.981	2.27	2.154	7.808	2.678	2.052
72	2.372	2.643	5.513	4.067	2.05	1.934	7.786	2.458	2.03
73	3.571	3.202	0.712	4.446	5.873	5.794	4.69	4.999	4.213
74	5.269	4.9	3.135	5.416	7.882	7.849	5.503	6.768	5.911
75	1.624	1.992	4.839	3.795	2.339	2.251	7.135	1.258	1.725
76	1.106	1.474	4.321	3.277	1.677	1.841	6.617	0.803	1.207
77	1.43	1.798	4.645	3.444	2.375	2.182	6.941	2.139	1.447
78	0.638	0.909	3.756	2.651	2.571	2.456	6.052	2.243	0.611
79	2.316	1.947	2.78	3.224	4.046	4.167	4.694	2.789	2.958
80	1.304	1.672	4.519	3.318	2.249	2.056	6.815	2.013	1.321
81	3.359	3.63	6.477	5.054	2.526	2.099	8.773	3.382	3.017
82	1.818	1.931	4.464	1.464	3.546	3.431	5.804	3.452	1.536
83	3.995	3.626	0.822	4.848	6.756	6.575	4.935	5.494	4.637
84	2.117	2.42	4.033	4.288	3.011	3.367	6.306	1.754	2.533

Table B.1 continued from previous page

85	1.566	1.869	3.938	3.737	2.642	2.998	6.211	1.384	1.982
86	2.549	2.82	5.69	4.362	2.539	2.423	7.963	2.947	2.207
87	1.015	1.318	4.188	3.186	2.741	2.683	6.461	1.747	1.231
88	3.496	3.127	0.735	4.219	6.278	6.075	4.282	5.396	4.138
89	2.267	1.898	1.043	3.158	5.049	4.846	3.315	4.167	2.909
90	1.315	1.683	4.53	3.329	1.849	1.402	6.826	1.356	1.332
91	1.096	1.464	4.311	3.11	2.457	2.264	6.607	1.805	1.113
92	1.41	1.779	4.626	3.28	2.506	2.281	6.922	2.292	1.165
93	2.086	2.199	4.714	2.298	3.787	3.672	6.614	3.753	2.101
94	2.883	3.251	5.997	4.925	2.514	2.616	8.27	1.328	2.928
95	3.589	3.957	6.804	5.58	1.226	1.744	9.1	2.342	3.634
96	3.444	3.806	6.653	5.137	1.375	0.857	8.926	2.524	3.401
97	2.765	2.878	5.355	1.857	4.601	4.486	6.835	4.507	2.591
98	3.981	3.612	0.92	4.872	6.763	6.56	4.943	5.85	4.623
99	2.645	3.014	5.855	4.338	2.182	1.749	8.127	2.596	2.602
100	3.145	3.514	6.355	4.838	2.331	1.813	8.627	3.096	3.102
101	3.227	3.595	6.332	5.269	2.769	2.994	8.605	1.515	3.272
102	2.391	2.759	5.496	4.562	1.901	2.244	7.769	0.718	2.436
103	2.669	3.037	5.806	4.84	2.179	2.522	8.079	0.91	2.746
104	3.521	3.889	6.626	5.563	3.031	3.288	8.899	1.762	3.566
105	2.188	2.557	5.404	3.121	2.884	2.769	7.437	2.85	2.04
106	0.949	1.318	4.165	2.991	2.045	1.93	6.461	2.011	0.813
107	1.821	2.092	4.962	3.408	2.843	2.511	7.235	2.516	1.479
108	2.172	2.443	5.313	3.759	2.492	2.376	7.586	2.867	1.83
109	1.913	2.184	5.054	3.5	2.751	2.603	7.327	2.608	1.571
110	2.156	1.787	1.685	3.047	4.938	4.735	3.958	4.007	2.798
111	4.655	4.768	7.245	3.747	6.382	6.267	8.725	6.327	4.373
112	2.527	2.83	4.369	4.698	3.059	3.253	6.642	1.801	2.906
113	3.259	3.628	6.469	4.952	3.251	2.818	8.741	3.533	3.215

Table B.1 continued from previous page

114	7.399	7.768	10.609	9.092	7.392	6.959	12.881	7.674	7.356
115	2.993	3.106	5.583	2.086	4.696	4.514	7.063	4.525	2.819
116	3.331	3.634	5.326	5.502	3.006	3.283	7.599	1.748	3.561
117	2.139	2.442	4.75	4.31	2.007	2.201	7.023	0.749	2.479
118	2.801	2.555	4.644	0.63	4.818	4.703	5.981	4.295	2.808
119	2.993	3.106	5.629	3.205	4.4	4.213	7.521	4.294	3.009
120	5.765	5.878	8.393	5.977	7.401	7.215	10.293	7.296	5.631
121	10.079	10.192	12.708	10.292	11.716	11.529	14.608	11.61	9.946
122	3.273	3.235	5.2	1.32	5.517	5.402	6.537	5.174	3.403
123	0.556	0.669	3.539	1.957	3.338	3.148	5.812	2.391	0.927
124	1.277	1.645	4.492	3.319	1.848	1.904	6.788	0.767	1.322
125	3.668	3.781	6.055	3.639	5.075	4.888	7.955	4.969	3.442
126	3.131	2.762	2.186	1.931	5.517	5.084	3.523	4.636	2.952
127	1.002	1.305	4.175	3.173	2.617	2.699	6.448	1.734	1.107
128	1.581	1.884	4.193	3.752	2.122	2.385	6.466	0.864	1.96
129	2.62	2.989	5.83	4.313	2.29	1.772	8.102	2.783	2.577
NODES	54	55	56	57	58	59	60	61	62
depot	6.88	6.356	5.777	5.636	5.803	6.831	1.424	4.809	4.861
1	3.262	2.248	1.567	1.447	2.652	2.723	5.683	0.62	0.672
2	3.453	1.21	1.301	1.548	3.643	1.49	6.776	1.263	0.679
3	4.343	2.1	2.191	2.438	4.491	2.115	6.688	2.153	1.569
4	5.042	2.52	2.89	3.137	5.232	2.045	8.365	2.852	2.268
5	4.195	3.371	2.659	2.552	3.487	3.846	5.215	1.693	1.984
6	4.477	8.044	7.579	6.546	6.6	7.607	12.45	7.237	7.566
7	7.935	7.386	6.722	6.615	6.121	7.861	4.228	5.756	5.984
8	5.25	8.833	8.265	7.331	7.388	8.358	13.239	8.01	8.339
9	4.224	1.705	2.076	2.323	4.418	1.23	7.504	2.038	1.454
10	4.527	2.009	2.379	2.627	4.721	1.534	7.807	2.341	1.758

Table B.1 continued from previous page

11	4.746	1.774	2.835	2.937	4.777	1.299	7.91	2.797	2.213
12	3.569	2.033	1.777	2.112	3.047	2.508	5.64	1.253	1.148
13	3.497	5.231	4.039	3.54	0.839	5.064	6.851	3.645	3.793
14	3.44	5.155	4.041	3.39	1.278	4.68	7.656	3.647	3.829
15	1.656	2.275	1.85	0.85	2.573	1.8	7.163	1.709	1.989
16	2.068	4.348	3.156	2.565	1.163	4.181	6.993	2.762	3.004
17	5.169	2.651	3.021	3.268	5.363	2.176	8.496	2.983	2.399
18	4.823	2.301	2.671	2.918	5.013	1.826	8.146	2.633	2.049
19	5.589	3.221	3.437	3.684	5.654	2.746	7.934	3.309	2.713
20	1.743	3.603	2.411	1.877	1.331	3.436	6.713	2.017	2.165
21	3.939	1.479	1.787	2.034	4.129	1.004	7.262	1.749	1.165
22	5.108	2.59	2.96	3.208	5.302	2.115	8.388	2.922	2.339
23	3.354	1.412	2.128	1.844	4.15	0.937	7.98	2.085	2.413
24	4.504	2.585	2.675	2.602	4.243	2.643	6.484	2.039	1.443
25	3.581	6.469	5.334	4.704	3.607	5.994	9.592	4.816	5.088
26	1.228	4.032	2.897	2.267	1.868	3.557	7.201	2.379	2.759
27	3.362	2.482	1.772	1.663	2.598	2.957	5.191	0.804	1.162
28	3.656	2.846	2.136	2.027	2.892	3.321	4.827	1.168	1.456
29	3.278	2.947	2.243	2.102	2.125	3.422	4.897	1.275	1.327
30	3.172	2.841	2.137	1.996	2.019	3.316	5.003	1.169	1.221
31	4.266	3.756	4.029	3.463	5.387	3.281	9.928	4.02	4.348
32	4.159	2.956	2.639	2.53	3.395	3.34	5.651	1.671	1.38
33	0.656	4.587	3.814	3.028	3.462	4.112	8.831	3.416	3.745
34	0.709	4.525	3.867	2.957	3.299	4.05	8.682	3.469	3.798
35	4.276	2.605	2.484	2.374	3.981	2.55	6.672	1.811	1.215
36	4.456	4.051	3.341	3.008	3.303	4.526	4.64	2.373	2.505
37	5.972	7.719	6.527	5.936	3.328	7.552	7.423	6.133	6.375
38	1.546	2.427	2.002	1.002	2.667	1.952	7.315	1.861	2.141
39	1.941	2.55	1.365	0.431	2.857	2.285	6.678	1.29	1.618

Table B.1 continued from previous page

40	4.411	3.602	2.892	2.783	3.647	4.077	4.812	1.924	2.214
41	2.316	4.064	2.872	2.281	0.717	3.897	6.305	2.471	2.523
42	7.907	11.474	10.906	9.972	10.029	11.037	15.88	10.667	10.996
43	0.584	4.502	3.742	2.943	3.377	4.027	8.746	3.344	3.673
44	3.036	2.998	2.294	2.153	1.862	3.473	5.16	1.326	1.378
45	2.475	1.875	0.683	1.144	2.669	2.35	6.28	0.285	0.614
46	2.571	2.178	0.766	0.816	2.782	2.368	6.649	0.654	0.982
47	4.76	3.676	3.51	2.944	5.652	3.201	9.49	3.501	3.829
48	0.724	3.294	2.66	1.726	2.881	2.819	7.973	2.456	2.785
49	4.921	3.616	3.466	3.356	4.157	3.63	6.273	2.669	2.197
50	4.806	3.679	3.362	3.253	4.042	3.986	5.84	2.394	2.103
51	6.1	5.59	5.863	5.297	7.221	5.115	11.762	5.797	6.125
52	4.376	2.359	2.449	2.482	4.123	2.373	6.555	1.919	1.323
53	2.911	2.291	1.325	1.522	2.387	2.766	6.237	0.663	0.715
54	0	3.931	3.158	2.372	2.806	3.456	8.175	2.76	3.089
55	3.931	0	2.198	2.119	4.54	0.475	7.673	2.16	1.889
56	3.158	2.198	0	1.48	3.348	2.673	6.963	0.968	1.297
57	2.372	2.119	1.48	0	2.849	2.594	6.854	0.859	1.187
58	2.806	4.54	3.348	2.849	0	4.373	7.022	2.954	3.102
59	3.456	0.475	2.673	2.594	4.373	0	8.148	2.635	2.169
60	8.175	7.673	6.963	6.854	7.022	8.148	0	5.995	6.224
61	2.76	2.16	0.968	0.859	2.954	2.635	5.995	0	0.624
62	3.089	1.889	1.297	1.187	3.102	2.169	6.224	0.624	0
63	3.244	2.689	2.025	1.734	2.746	3.164	5.375	1.058	1.39
64	3.368	2.234	1.576	1.955	2.916	2.709	5.509	1.122	1.18
65	6.112	9.695	9.056	8.193	8.25	9.22	14.101	8.872	9.201
66	2.658	3.103	2.358	2.249	1.63	3.578	6.183	1.391	1.565
67	3.576	7.158	6.519	5.656	5.286	6.683	11.271	6.336	6.665
68	4.298	7.818	6.683	6.053	4.956	7.343	10.941	6.057	6.437

Table B.1 continued from previous page

69	4.622	2.104	2.474	2.721	4.816	1.629	7.902	2.436	1.852
70	3.927	3.632	2.968	2.781	3.161	4.107	4.581	2.001	2.095
71	3.892	3.741	3.077	2.906	2.718	4.216	4.576	2.126	2.194
72	4.242	3.719	3.055	2.659	3.068	4.194	4.356	2.088	2.353
73	5.092	3.177	3.866	3.37	5.844	2.702	9.395	3.856	4.184
74	6.077	4.875	5.564	5.068	7.184	4.4	11.52	5.554	5.882
75	4.099	2.605	2.307	2.197	4.112	3.08	6.744	1.634	1.182
76	3.581	2.087	1.789	1.679	3.594	2.562	6.424	1.116	0.862
77	3.665	2.392	2.113	2.003	3.693	2.77	6.64	1.44	0.816
78	3.113	2.195	1.321	0.925	2.954	2.456	6.074	0.354	0.978
79	3.87	0.896	2.644	2.36	4.191	0.421	7.768	2.601	2.59
80	3.622	2.266	1.987	1.877	3.67	2.741	6.514	1.314	0.69
81	5.332	4.706	4.042	3.646	4.179	5.181	4.568	3.075	3.364
82	1.389	3.578	2.497	1.998	1.417	3.103	6.8	1.871	2.251
83	5.509	3.601	4.29	3.766	6.408	3.126	10.246	4.28	4.608
84	4.592	2.07	2.44	2.687	4.782	1.595	7.915	2.402	1.818
85	4.041	1.798	1.889	2.136	4.231	1.356	7.364	1.851	1.267
86	4.287	3.896	3.232	3.042	3.425	4.371	4.845	2.265	2.359
87	3.486	1.386	1.338	2.037	3.581	1.861	6.424	1.3	1.168
88	4.865	3.715	3.79	3.224	5.909	3.24	9.676	3.781	4.109
89	3.804	2.727	2.561	1.995	4.68	2.252	8.447	2.552	2.88
90	3.79	2.277	1.998	1.888	3.582	2.752	5.823	1.325	0.701
91	3.414	2.058	1.779	1.669	3.462	2.533	6.306	1.106	0.482
92	3.191	2.787	2.093	1.952	2.407	3.262	5.113	1.125	1.177
93	2.209	3.938	2.765	2.173	1.133	3.463	6.609	2.371	2.612
94	5.358	3.687	3.566	3.456	5.063	3.632	7.232	2.893	2.297
95	5.891	4.489	4.272	4.162	5.326	4.504	6.996	3.512	3.003
96	5.506	4.444	4.127	4.018	4.353	4.828	5.158	3.159	2.868
97	1.691	4.579	3.444	2.814	1.872	4.104	7.702	2.926	3.306

Table B.1 continued from previous page

98	5.518	3.957	4.275	3.709	6.394	3.482	10.089	4.266	4.594
99	4.848	4.038	3.328	3.219	3.897	4.513	4.702	2.36	2.648
100	5.046	4.538	3.828	3.719	3.893	5.013	4.698	2.86	3.095
101	5.702	3.83	3.91	3.8	5.441	3.797	7.61	3.237	2.641
102	4.866	3.033	3.074	2.964	4.605	2.961	6.774	2.401	1.805
103	5.144	3.225	3.352	3.242	4.883	3.239	7.052	2.679	2.083
104	5.996	4.077	4.204	4.094	5.735	4.091	7.904	3.531	2.935
105	3.032	3.502	2.871	2.73	1.433	3.977	5.737	1.903	1.955
106	3.043	2.502	1.632	1.491	2.479	2.885	5.508	0.664	0.716
107	3.319	3.168	2.504	2.396	2.145	3.643	5.082	1.569	1.621
108	3.67	3.519	2.855	2.747	2.496	3.994	4.731	1.92	1.972
109	3.411	3.26	2.596	2.488	2.237	3.735	4.99	1.661	1.713
110	3.693	2.616	2.45	1.884	4.569	2.141	8.264	2.441	2.769
111	3.581	6.469	5.334	4.704	3.607	5.994	9.592	4.816	5.088
112	4.998	2.48	2.85	3.097	5.192	2.005	7.941	2.812	2.228
113	5.153	4.652	3.942	3.833	4	5.127	4.689	2.974	3.202
114	9.294	8.792	8.082	7.973	8.141	9.267	1.119	7.114	7.343
115	1.92	4.807	3.672	3.042	1.679	4.332	7.509	3.154	3.521
116	5.806	3.437	3.654	3.901	5.73	2.962	7.899	3.526	2.93
117	4.61	2.371	2.462	2.709	4.648	2.385	6.817	2.424	1.84
118	0.464	3.846	3.29	2.31	2.632	3.371	7.965	3.086	3.415
119	3.116	4.853	3.672	3.081	0.71	4.378	7.105	3.271	3.323
120	5.888	7.617	6.444	5.852	3.244	7.142	7.268	6.05	6.291
121	10.203	11.932	10.758	10.167	7.559	11.457	11.582	10.364	10.606
122	0.798	4.402	3.899	2.866	3.388	3.927	8.771	3.558	3.887
123	2.637	2.427	1.235	1.125	2.113	2.887	6.749	0.841	1.169
124	3.623	2.258	1.96	1.85	3.671	2.733	6.253	1.287	0.691
125	3.55	5.279	4.347	3.514	1.402	4.804	7.657	3.779	3.953
126	2.577	2.743	3.448	2.415	3.698	2.268	8.761	3.274	3.554

Table B.1 continued from previous page

127	3.473	1.373	1.325	2.024	3.457	1.848	6.3	1.287	1.155
128	4.052	1.813	1.904	2.151	4.246	1.828	7.001	1.866	1.282
129	4.591	4.013	3.303	3.194	3.438	4.488	4.243	2.335	2.623
NODES	63	64	65	66	67	68	69	70	71
depot	4.534	4.668	12.806	4.888	10.015	9.685	6.632	3.773	3.529
1	0.779	0.508	9.188	1.022	6.438	6.073	2.524	1.548	1.653
2	1.792	1.337	9.565	2.206	7.029	6.978	1.174	2.735	2.86
3	2.103	2.143	10.455	3.03	7.919	7.868	1.318	3.007	3.002
4	3.381	2.926	11.154	3.795	8.618	8.567	0.727	4.324	4.449
5	1.073	1.207	10.107	2	7.321	6.991	3.228	1.534	1.529
6	7.517	7.521	1.651	6.919	1.889	1.644	9.099	8.188	8.167
7	5.136	5.27	13.861	5.943	11.031	10.701	7.663	4.341	4.336
8	8.203	8.31	1.9	7.708	2.678	2.432	9.872	8.977	8.956
9	2.567	2.112	10.336	2.981	7.8	7.753	0.473	3.51	3.619
10	2.87	2.415	10.639	3.284	8.103	8.056	0.095	3.814	3.939
11	2.973	2.471	10.513	3.378	7.976	8.112	1.047	3.908	4.007
12	0.703	0.201	9.373	1.602	6.836	6.592	2.309	1.905	1.954
13	3.426	3.607	8.941	2.321	5.977	5.647	5.507	3.841	3.398
14	3.38	3.562	6.972	2.264	4.008	3.678	5.509	3.795	3.352
15	1.788	1.825	7.458	2.281	4.94	5.543	3.423	3.33	3.259
16	2.661	2.725	7.242	1.439	4.278	3.948	4.624	3.132	2.689
17	3.512	3.057	11.281	3.926	8.745	8.698	0.858	4.455	4.58
18	3.162	2.707	10.935	3.576	8.399	8.348	0.508	4.105	4.23
19	3.169	3.306	11.701	4.096	9.165	9.087	1.196	4.073	4.068
20	2.036	2.242	7.822	1.281	4.868	4.538	3.879	2.651	2.431
21	2.278	1.823	10.051	2.692	7.515	7.464	1.179	3.221	3.346
22	3.451	2.996	11.22	3.865	8.684	8.637	0.796	4.394	4.503
23	2.716	2.595	9.175	3.475	6.638	7.188	2.468	3.987	4.096

Table B.1 continued from previous page

24	2.049	2.056	10.538	2.826	8.001	7.817	1.776	2.803	2.798
25	4.96	5.069	4.643	4.038	1.679	1.349	6.802	5.483	5.288
26	2.523	2.705	7.08	1.671	4.116	3.786	4.365	3.046	2.917
27	0.254	0.318	9.288	1.181	6.502	6.172	2.711	1.456	1.505
28	0.548	0.682	9.582	1.475	6.796	6.466	3.075	1.146	1.141
29	0.833	1.015	9.204	1.286	6.374	6.044	3.179	1.423	1.079
30	0.727	0.909	9.098	1.18	6.268	5.938	3.073	1.317	0.973
31	4.647	4.543	10.033	5.215	7.55	8.24	4.432	5.922	6.031
32	1.051	1.185	10.085	1.978	7.299	6.969	2.474	1.97	1.965
33	3.9	4.024	6.397	3.314	3.861	4.583	5.278	4.583	4.548
34	3.829	3.964	5.821	3.151	3.285	4.007	5.331	4.42	4.399
35	2.01	1.942	10.31	2.564	7.773	7.555	1.684	2.541	2.536
36	1.753	1.887	10.382	2.464	7.552	7.222	4.173	0.861	0.797
37	5.915	6.096	11.237	4.81	8.273	7.943	7.995	5.995	5.552
38	1.94	1.977	7.338	2.433	4.83	5.544	3.575	3.482	3.411
39	1.303	1.875	7.762	2.565	5.225	5.775	3.135	2.895	2.992
40	1.374	1.438	10.337	2.23	7.551	7.221	3.459	1.494	1.489
41	2.029	2.211	7.93	1.155	4.966	4.636	4.34	2.444	2.001
42	10.844	10.95	3.967	10.349	5.319	5.073	12.529	11.618	11.597
43	3.815	3.952	6.683	3.229	4.147	4.869	5.193	4.498	4.476
44	0.884	1.066	8.962	1.206	6.111	5.781	3.201	1.299	0.856
45	1.342	0.893	8.587	1.675	6.051	6.004	2.151	2.285	2.394
46	1.613	1.262	8.392	2.044	5.855	6.117	2.454	2.556	2.665
47	4.122	4.105	10.531	4.892	8.044	8.594	3.993	5.426	5.535
48	2.598	2.733	6.65	2.733	4.124	4.846	4.322	4.002	3.981
49	2.131	2.265	10.847	2.74	8.061	7.731	2.764	2.275	2.27
50	1.698	1.908	10.732	2.625	7.946	7.616	3.12	2.159	2.154
51	6.395	6.377	11.868	7.073	9.384	10.074	6.266	7.699	7.808
52	1.779	1.843	10.418	2.706	7.881	7.676	1.506	2.683	2.678

Table B.1 continued from previous page

53	1	1.134	8.734	1.057	6.052	5.722	2.53	1.943	2.052
54	3.244	3.368	6.112	2.658	3.576	4.298	4.622	3.927	3.892
55	2.689	2.234	9.695	3.103	7.158	7.818	2.104	3.632	3.741
56	2.025	1.576	9.056	2.358	6.519	6.683	2.474	2.968	3.077
57	1.734	1.955	8.193	2.249	5.656	6.053	2.721	2.781	2.906
58	2.746	2.916	8.25	1.63	5.286	4.956	4.816	3.161	2.718
59	3.164	2.709	9.22	3.578	6.683	7.343	1.629	4.107	4.216
60	5.375	5.509	14.101	6.183	11.271	10.941	7.902	4.581	4.576
61	1.058	1.122	8.872	1.391	6.336	6.057	2.436	2.001	2.126
62	1.39	1.18	9.201	1.565	6.665	6.437	1.852	2.095	2.194
63	0	0.572	9.065	1.317	6.528	6.308	2.965	1.592	1.689
64	0.572	0	9.172	1.401	6.635	6.391	2.51	1.774	1.823
65	9.065	9.172	0	8.57	3.54	3.294	10.734	9.839	9.818
66	1.317	1.401	8.57	0	5.717	5.387	3.379	1.926	2.062
67	6.528	6.635	3.54	5.717	0	1.238	8.198	7.162	6.945
68	6.308	6.391	3.294	5.387	1.238	0	8.151	6.832	6.637
69	2.965	2.51	10.734	3.379	8.198	8.151	0	3.909	4.034
70	1.592	1.774	9.839	1.926	7.162	6.832	3.909	0	0.443
71	1.689	1.823	9.818	2.062	6.945	6.637	4.034	0.443	0
72	1.469	1.603	10.168	2.38	7.295	6.987	3.824	0.577	0.448
73	4.454	3.886	10.913	4.955	8.376	8.721	4.325	5.487	5.612
74	6.152	6.135	11.854	6.944	9.318	10.041	6.023	7.456	7.565
75	2.069	1.567	10.211	2.704	7.675	7.447	2.495	2.782	3.103
76	1.747	1.247	9.693	2.057	7.157	6.929	2.169	2.372	2.686
77	1.703	1.201	9.577	2.296	6.9	6.57	2.157	2.713	2.934
78	1.148	1.03	9.118	1.324	6.581	6.267	2.159	1.856	1.981
79	2.831	2.895	9.641	3.758	7.104	7.068	2.044	4.033	4.082
80	1.577	1.075	9.369	2.17	7.027	6.589	2.284	2.587	2.808
81	2.456	2.59	11.258	3.34	8.428	8.098	4.811	1.738	1.733

Table B.1 continued from previous page

82	2.123	2.206	7.301	1.269	4.624	4.294	3.965	2.538	2.517
83	4.878	4.861	11.286	5.67	8.75	9.35	4.749	6.182	6.291
84	2.932	2.476	10.704	3.346	8.168	8.117	0.57	3.875	3.949
85	2.38	1.925	10.153	2.794	7.617	7.566	0.743	3.273	3.398
86	1.856	2.038	10.199	2.19	7.513	7.183	4.173	0.36	0.707
87	1.487	0.985	9.598	2.081	7.062	6.916	1.662	2.61	2.719
88	4.395	4.291	10.686	5.171	8.149	8.839	4.728	5.683	5.792
89	3.166	3.062	9.609	3.942	7.088	7.638	3.783	4.454	4.563
90	1.238	1.086	9.796	2.165	7.259	7.054	2.553	2.142	2.137
91	1.369	0.867	9.526	1.962	6.99	6.797	2.334	2.491	2.6
92	0.946	0.994	9.117	1.199	6.331	6.001	3.029	0.922	1.047
93	2.333	2.448	8.135	1.047	5.22	4.89	4.233	2.589	2.146
94	2.719	2.682	11.392	3.646	8.855	8.637	2.766	3.623	3.618
95	2.982	3.276	11.803	3.697	8.932	8.602	3.731	3.501	3.496
96	2.539	2.673	11.432	3.466	8.602	8.272	3.962	1.912	1.907
97	3.07	3.152	6.789	2.148	3.825	3.495	4.912	3.593	3.376
98	4.88	4.776	11.323	5.656	8.802	9.352	4.913	6.168	6.277
99	1.74	1.874	10.774	2.667	7.988	7.658	4.034	1.456	1.451
100	2.24	2.374	10.972	3.054	8.142	7.812	4.534	1.452	1.447
101	3.097	3.073	11.736	4.024	9.199	8.994	3.021	4.001	3.996
102	2.261	2.418	10.9	3.188	8.363	8.158	2.198	3.165	3.16
103	2.539	2.468	11.178	3.466	8.641	8.457	2.416	3.443	3.438
104	3.391	3.322	12.03	4.318	9.493	9.288	3.268	4.295	4.29
105	1.461	1.643	8.646	1.463	5.682	5.352	3.778	1.876	1.433
106	0.674	0.552	8.955	0.849	6.278	5.839	2.568	1.379	1.478
107	1.127	1.309	9.245	1.489	6.394	6.064	3.473	1.016	0.573
108	1.478	1.66	9.596	1.84	6.723	6.415	3.824	0.665	0.222
109	1.219	1.401	9.337	1.581	6.464	6.156	3.565	0.924	0.481
110	3.055	2.951	9.498	3.831	6.977	7.527	3.672	4.343	4.452

Table B.1 continued from previous page

111	4.96	5.069	4.643	4.038	1.679	1.349	6.802	5.483	5.288
112	3.162	2.66	11.11	3.755	8.574	8.527	0.687	4.26	4.255
113	2.354	2.488	11.079	3.161	8.249	7.919	4.881	1.559	1.554
114	6.494	6.628	15.22	7.302	12.39	12.06	9.021	5.7	5.695
115	3.178	3.242	7.093	1.956	4.129	3.799	5.14	3.648	3.205
116	3.386	3.315	11.918	4.313	9.382	9.283	1.413	4.29	4.285
117	2.304	2.233	10.722	3.231	8.186	8.139	1.239	3.208	3.203
118	3.228	3.285	6.4	2.435	3.864	4.466	4.952	3.81	3.681
119	2.829	3.011	8.73	1.955	5.766	5.436	5.14	3.244	2.801
120	5.831	6.013	8.612	4.726	5.648	5.318	7.912	6.246	5.803
121	10.146	10.327	15.468	9.041	12.504	12.174	12.226	10.226	9.783
122	3.837	3.841	5.505	3.24	2.968	3.691	5.42	4.509	4.488
123	1.813	1.364	8.424	1.821	5.778	5.448	2.703	2.763	2.888
124	1.578	1.076	9.735	2.228	7.199	7.006	2.205	2.543	2.639
125	3.504	3.686	7.328	2.388	4.364	4.034	5.767	3.591	3.148
126	3.386	3.39	8.398	3.55	5.861	6.575	3.799	4.819	4.798
127	1.363	0.861	9.585	1.957	7.049	6.792	1.649	2.486	2.595
128	2.216	1.714	10.164	2.809	7.628	7.581	1.146	3.338	3.387
129	1.715	1.849	10.517	2.599	7.687	7.357	4.221	0.997	0.992
NODES	72	73	74	75	76	77	78	79	80
depot	3.548	8.342	10.351	5.32	5	5.677	4.711	6.927	5.551
1	1.807	4.153	6.152	1.682	1.164	1.488	0.522	3.144	1.362
2	2.87	4.115	5.89	1.708	1.19	1.495	0.985	1.911	1.369
3	2.782	4.765	6.51	1.558	1.127	2.385	1.875	2.531	2.259
4	4.459	4.741	6.439	2.91	2.779	2.572	2.574	2.46	2.699
5	1.309	5.093	7.218	2.132	1.722	2.063	1.831	3.466	1.937
6	8.517	9.333	10.204	8.576	8.058	7.926	7.471	8.028	8.053
7	4.116	9.156	11.281	6.577	6.207	6.401	5.834	7.529	6.275

Table B.1 continued from previous page

8	9.306	10.051	10.992	9.349	8.831	8.715	8.256	8.779	8.842
9	3.598	3.926	5.624	2.096	1.965	1.758	1.76	1.645	1.885
10	3.901	4.23	5.928	2.4	2.264	2.062	2.064	1.949	2.189
11	4.004	3.995	5.693	2.598	2.55	2.442	2.519	1.714	2.503
12	1.734	4.087	6.336	1.366	1.046	1	1.231	2.929	0.874
13	3.748	6.535	7.875	4.803	4.285	4.384	3.645	4.882	4.361
14	3.702	6.263	7.818	4.855	4.321	4.327	3.588	4.825	4.363
15	3.257	3.526	5.186	2.94	2.481	2.574	1.775	2.221	2.448
16	3.039	5.438	6.992	3.97	3.496	3.501	2.763	3.999	3.478
17	4.59	4.872	6.57	3.041	2.91	2.703	2.705	2.591	2.83
18	4.24	4.522	6.22	2.691	2.56	2.353	2.355	2.241	2.48
19	3.848	5.442	7.14	2.721	2.193	3.273	3.121	3.161	3.4
20	2.781	4.948	6.144	3.175	2.657	2.797	2.061	3.295	2.733
21	3.356	3.629	5.404	2.194	1.676	1.958	1.471	1.425	1.855
22	4.482	4.811	6.509	2.981	2.849	2.643	2.645	2.53	2.77
23	4.074	2.586	4.165	3.423	2.905	3.229	2.34	0.516	3.103
24	2.578	5.119	7.038	1.471	0.923	2.259	2.393	3.059	2.133
25	5.638	7.372	8.712	6.098	5.58	5.221	4.918	5.719	5.348
26	3.267	4.935	6.275	3.711	3.157	2.784	2.481	3.282	2.911
27	1.285	4.204	6.33	1.815	1.493	1.449	1.012	2.577	1.323
28	0.921	4.568	6.693	1.989	1.665	1.813	1.306	2.941	1.687
29	1.429	4.808	6.817	2.468	1.819	2.067	1.177	3.274	1.941
30	1.323	4.702	6.711	2.362	1.713	1.961	1.071	3.168	1.835
31	6.009	2.856	3.669	5.358	4.84	5.164	4.275	2.86	5.038
32	1.745	5.071	7.196	1.528	1.118	1.459	1.809	3.444	1.333
33	4.898	5.748	6.733	4.755	4.237	4.321	3.769	4.526	4.278
34	4.749	5.677	6.647	4.808	4.29	4.158	3.822	4.455	4.285
35	2.316	4.891	6.9	1.357	0.695	2.031	2.165	2.966	1.905
36	0.349	5.773	7.892	3.194	2.825	3.018	2.355	4.146	2.892

Table B.1 continued from previous page

37	5.902	8.809	10.364	7.341	6.867	6.873	6.134	7.371	6.849
38	3.409	3.678	5.066	3.092	2.633	2.726	1.927	2.373	2.6
39	2.772	3.151	4.849	2.628	2.11	2.434	1.356	1.929	2.308
40	1.269	5.324	7.449	2.362	1.952	2.293	2.061	3.697	2.167
41	2.351	5.368	6.708	3.664	3.015	3.217	2.373	3.715	3.137
42	11.947	12.692	13.633	12.006	11.488	11.356	10.897	11.458	11.483
43	4.826	5.663	6.648	4.683	4.165	3.65	3.697	4.441	4.192
44	1.206	4.859	6.868	2.519	1.87	2.118	1.228	3.325	1.992
45	2.372	3.571	5.269	1.624	1.106	1.43	0.638	2.316	1.304
46	2.643	3.202	4.9	1.992	1.474	1.798	0.909	1.947	1.672
47	5.513	0.712	3.135	4.839	4.321	4.645	3.756	2.78	4.519
48	4.067	4.446	5.416	3.795	3.277	3.444	2.651	3.224	3.318
49	2.05	5.873	7.882	2.339	1.677	2.375	2.571	4.046	2.249
50	1.934	5.794	7.849	2.251	1.841	2.182	2.456	4.167	2.056
51	7.786	4.69	5.503	7.135	6.617	6.941	6.052	4.694	6.815
52	2.458	4.999	6.768	1.258	0.803	2.139	2.243	2.789	2.013
53	2.03	4.213	5.911	1.725	1.207	1.447	0.611	2.958	1.321
54	4.242	5.092	6.077	4.099	3.581	3.665	3.113	3.87	3.622
55	3.719	3.177	4.875	2.605	2.087	2.392	2.195	0.896	2.266
56	3.055	3.866	5.564	2.307	1.789	2.113	1.321	2.644	1.987
57	2.659	3.37	5.068	2.197	1.679	2.003	0.925	2.36	1.877
58	3.068	5.844	7.184	4.112	3.594	3.693	2.954	4.191	3.67
59	4.194	2.702	4.4	3.08	2.562	2.77	2.456	0.421	2.741
60	4.356	9.395	11.52	6.744	6.424	6.64	6.074	7.768	6.514
61	2.088	3.856	5.554	1.634	1.116	1.44	0.354	2.601	1.314
62	2.353	4.184	5.882	1.182	0.862	0.816	0.978	2.59	0.69
63	1.469	4.454	6.152	2.069	1.747	1.703	1.148	2.831	1.577
64	1.603	3.886	6.135	1.567	1.247	1.201	1.03	2.895	1.075
65	10.168	10.913	11.854	10.211	9.693	9.577	9.118	9.641	9.369

Table B.1 continued from previous page

66	2.38	4.955	6.944	2.704	2.057	2.296	1.324	3.758	2.17
67	7.295	8.376	9.318	7.675	7.157	6.9	6.581	7.104	7.027
68	6.987	8.721	10.041	7.447	6.929	6.57	6.267	7.068	6.589
69	3.824	4.325	6.023	2.495	2.169	2.157	2.159	2.044	2.284
70	0.577	5.487	7.456	2.782	2.372	2.713	1.856	4.033	2.587
71	0.448	5.612	7.565	3.103	2.686	2.934	1.981	4.082	2.808
72	0	5.489	7.543	2.91	2.541	2.734	2.115	3.862	2.608
73	5.489	0	2.844	4.714	4.196	4.741	3.631	3.102	4.874
74	7.543	2.844	0	6.723	6.205	6.698	5.64	4.681	6.572
75	2.91	4.714	6.723	0	0.662	1.998	1.988	3.156	1.872
76	2.541	4.196	6.205	0.662	0	1.495	1.47	2.983	1.369
77	2.734	4.741	6.698	1.998	1.495	0	1.11	2.818	0.542
78	2.115	3.631	5.64	1.988	1.47	1.11	0	2.856	1.652
79	3.862	3.102	4.681	3.156	2.983	2.818	2.856	0	3.162
80	2.608	4.874	6.572	1.872	1.369	0.542	1.652	3.162	0
81	1.513	6.352	8.53	3.459	3.314	3.23	2.927	4.848	3.357
82	2.867	4.427	5.791	3.261	2.649	2.276	1.973	2.774	2.403
83	6.269	1.379	3.417	5.618	5.1	5.281	4.512	2.797	5.298
84	4.009	4.297	5.995	2.121	2.33	1.965	2.124	2.01	2.092
85	3.458	4.058	5.756	1.752	1.778	1.414	1.573	1.772	1.541
86	0.841	5.748	7.72	2.961	2.732	2.42	2.117	4.297	2.547
87	2.518	4.52	6.218	1.582	1.064	1.396	1.474	2.282	1.27
88	5.77	0.538	2.4	5.119	4.601	4.925	4.036	3.06	4.799
89	4.541	1.375	3.073	3.89	3.372	3.696	2.807	1.831	3.57
90	1.917	4.865	6.583	1.079	0.669	1.057	1.679	3.173	0.931
91	2.4	4.666	6.364	1.664	1.344	0.334	1.444	2.954	0.208
92	1.181	4.658	6.667	2.308	1.669	1.993	1.027	3.269	1.867
93	2.496	5.046	6.601	3.579	3.104	3.11	2.371	3.608	3.217
94	3.398	5.973	7.982	2.439	1.777	3.113	3.247	4.048	2.987

Table B.1 continued from previous page

95	3.276	6.679	8.688	2.803	2.483	3.391	3.414	4.92	3.265
96	1.687	6.559	8.684	3.016	2.606	2.947	3.297	4.932	2.821
97	3.726	5.482	6.822	4.258	3.704	3.331	3.028	3.829	3.458
98	6.255	1.023	3.061	5.604	5.086	5.41	4.521	3.153	5.284
99	1.231	5.76	7.885	2.938	2.528	2.869	2.498	4.133	2.743
100	1.227	6.26	8.385	3.438	3.028	3.369	2.945	4.633	3.243
101	3.776	6.317	8.197	2.669	2.121	3.457	3.591	4.218	3.331
102	2.94	5.481	7.361	1.833	1.285	2.621	2.755	3.382	2.495
103	3.218	5.759	7.639	2.111	1.563	2.899	3.033	3.66	2.773
104	4.07	6.611	8.491	2.963	2.415	3.751	3.885	4.512	3.625
105	1.783	5.436	7.424	3.096	2.447	2.695	1.805	3.902	2.569
106	1.637	4.197	6.206	1.87	1.208	1.532	0.566	3.063	1.406
107	0.923	5.102	6.992	2.53	2.113	2.361	1.471	3.568	2.235
108	0.572	5.453	7.343	2.881	2.464	2.712	1.822	3.919	2.586
109	0.831	5.194	7.084	2.622	2.205	2.453	1.563	3.66	2.327
110	4.43	2.017	3.715	3.779	3.261	3.585	2.696	1.72	3.459
111	5.638	7.372	8.712	6.098	5.58	5.221	4.918	5.719	5.348
112	4.035	4.701	6.399	2.87	2.38	2.532	2.534	2.42	2.659
113	1.334	6.374	8.499	3.795	3.425	3.619	3.052	4.747	3.493
114	5.475	10.514	12.639	7.863	7.543	7.759	7.193	8.887	7.633
115	3.555	5.71	7.05	4.486	3.932	3.559	3.256	4.057	3.686
116	4.065	5.658	7.356	2.937	2.41	3.489	3.338	3.377	3.616
117	2.983	5.011	6.78	1.759	1.328	2.656	2.146	2.801	2.53
118	4.031	5.076	5.968	4.019	3.699	3.548	3.235	3.792	3.675
119	3.151	5.961	7.508	4.464	3.815	4.017	3.173	4.515	3.937
120	6.153	8.725	10.28	7.258	6.783	6.789	6.05	7.287	6.896
121	10.133	13.04	14.595	11.572	11.098	11.104	10.365	11.602	11.08
122	4.838	5.653	6.524	4.897	4.379	4.247	3.791	4.348	4.374
123	2.843	3.871	5.569	2.115	1.661	1.985	0.907	2.616	1.859

Table B.1 continued from previous page

124	2.419	4.367	6.376	0.491	0.171	1.507	1.641	3.154	1.381
125	3.498	6.387	7.942	5.053	4.445	4.451	3.712	4.949	4.558
126	4.855	2.732	3.51	4.564	4.046	4.25	3.34	1.847	4.124
127	2.394	4.507	6.205	1.458	0.94	1.52	1.461	2.269	1.394
128	3.167	4.454	6.223	1.976	1.512	2.098	1.588	2.244	1.972
129	0.772	5.735	7.86	3.125	2.715	2.98	2.473	4.108	2.854
NODES	81	82	83	84	85	86	87	88	89
depot	3.56	5.505	9.021	6.598	6.047	4.037	5.107	8.59	7.361
1	2.779	1.887	4.878	2.49	1.939	1.812	1.493	4.379	3.15
2	3.809	2.792	4.616	1.14	0.588	2.999	0.489	4.407	3.178
3	3.682	3.682	5.236	1.496	1.126	3.271	1.379	5.106	3.86
4	5.398	4.381	5.165	0.986	1.158	4.588	2.078	5.144	4.049
5	1.91	2.806	5.944	3.475	3.062	1.798	2.122	5.374	4.145
6	9.607	5.65	9.636	9.069	8.518	8.548	7.963	9.035	7.958
7	4.986	6.56	10.007	7.629	7.077	4.605	6.185	9.437	8.208
8	10.396	6.439	10.424	9.842	9.291	9.337	8.736	9.824	8.747
9	4.584	3.567	4.35	0.439	0.344	3.774	1.254	4.329	3.476
10	4.847	3.87	4.654	0.475	0.648	4.078	1.567	4.633	3.688
11	4.991	3.926	4.419	0.477	1.209	4.172	1.734	4.398	3.545
12	2.721	2.407	5.062	2.275	1.724	2.169	0.784	4.492	3.263
13	4.859	2.108	7.099	5.473	4.922	4.105	4.272	6.6	5.371
14	4.813	2.051	6.687	5.475	4.924	4.059	4.274	6.204	4.975
15	4.244	1.36	3.95	3.395	2.986	3.594	2.29	3.577	2.151
16	4.15	1.225	5.862	4.59	4.039	3.396	3.389	5.379	4.15
17	5.529	4.512	5.296	1.117	1.289	4.719	2.209	5.317	4.18
18	5.179	4.162	4.946	0.767	0.939	4.369	1.859	4.998	3.83
19	4.835	4.842	5.866	1.687	1.859	4.337	2.625	5.846	4.282
20	3.87	0.521	5.471	3.845	3.294	2.915	2.644	4.922	3.743

Table B.1 continued from previous page

21	4.295	3.278	4.13	1.224	0.544	3.485	0.975	3.97	3.158
22	5.469	4.451	5.235	1.056	1.229	4.658	2.148	5.214	4.358
23	5.061	3.058	3.01	2.434	2.196	4.251	2.749	2.544	1.315
24	3.496	3.572	5.764	1.979	1.654	3.067	1.863	5.524	4.295
25	6.749	2.945	8.001	6.768	6.217	5.834	5.567	7.49	6.289
26	4.358	0.508	5.564	4.331	3.78	3.397	3.13	5.077	3.852
27	2.272	1.987	5.056	2.724	2.173	1.72	1.233	4.486	3.257
28	1.908	2.281	5.419	3.088	2.537	1.41	1.597	4.849	3.62
29	2.054	1.903	5.555	3.145	2.594	1.687	1.852	5.056	3.827
30	2.16	1.797	5.449	3.039	2.488	1.581	1.746	4.95	3.721
31	6.996	3.97	3.101	4.579	4.54	6.186	4.673	2.448	1.481
32	2.346	2.784	5.922	2.721	2.352	2.234	1.96	5.352	4.123
33	5.988	2.045	6.165	5.248	4.697	4.943	4.142	5.521	4.46
34	5.839	1.882	6.079	5.301	4.75	4.78	4.195	5.45	4.389
35	3.275	3.344	5.671	1.921	1.562	2.805	1.759	5.296	4.067
36	1.797	3.081	6.618	4.293	3.742	1.125	2.802	6.054	4.825
37	7.285	4.597	9.233	7.961	7.41	6.259	6.76	8.75	7.521
38	4.396	1.25	4.102	3.547	3.138	3.746	2.442	3.729	2.303
39	3.759	1.645	3.575	3.118	2.567	3.159	2.002	3.092	1.863
40	1.362	3.036	6.175	3.706	3.293	1.758	2.353	5.605	4.376
41	3.462	0.941	5.932	4.306	3.755	2.708	3.048	5.433	4.204
42	13.037	9.08	13.065	12.499	11.948	11.978	11.393	12.465	11.388
43	5.903	1.96	6.08	5.176	4.625	4.858	4.057	5.436	4.375
44	2.317	1.661	5.606	3.196	2.645	1.563	1.903	5.107	3.878
45	3.359	1.818	3.995	2.117	1.566	2.549	1.015	3.496	2.267
46	3.63	1.931	3.626	2.42	1.869	2.82	1.318	3.127	1.898
47	6.477	4.464	0.822	4.033	3.938	5.69	4.188	0.735	1.043
48	5.054	1.464	4.848	4.288	3.737	4.362	3.186	4.219	3.158
49	2.526	3.546	6.756	3.011	2.642	2.539	2.741	6.278	5.049

Table B.1 continued from previous page

50	2.099	3.431	6.575	3.367	2.998	2.423	2.683	6.075	4.846
51	8.773	5.804	4.935	6.306	6.211	7.963	6.461	4.282	3.315
52	3.382	3.452	5.494	1.754	1.384	2.947	1.747	5.396	4.167
53	3.017	1.536	4.637	2.533	1.982	2.207	1.231	4.138	2.909
54	5.332	1.389	5.509	4.592	4.041	4.287	3.486	4.865	3.804
55	4.706	3.578	3.601	2.07	1.798	3.896	1.386	3.715	2.727
56	4.042	2.497	4.29	2.44	1.889	3.232	1.338	3.79	2.561
57	3.646	1.998	3.766	2.687	2.136	3.042	2.037	3.224	1.995
58	4.179	1.417	6.408	4.782	4.231	3.425	3.581	5.909	4.68
59	5.181	3.103	3.126	1.595	1.356	4.371	1.861	3.24	2.252
60	4.568	6.8	10.246	7.915	7.364	4.845	6.424	9.676	8.447
61	3.075	1.871	4.28	2.402	1.851	2.265	1.3	3.781	2.552
62	3.364	2.251	4.608	1.818	1.267	2.359	1.168	4.109	2.88
63	2.456	2.123	4.878	2.932	2.38	1.856	1.487	4.395	3.166
64	2.59	2.206	4.861	2.476	1.925	2.038	0.985	4.291	3.062
65	11.258	7.301	11.286	10.704	10.153	10.199	9.598	10.686	9.609
66	3.34	1.269	5.67	3.346	2.794	2.19	2.081	5.171	3.942
67	8.428	4.624	8.75	8.168	7.617	7.513	7.062	8.149	7.088
68	8.098	4.294	9.35	8.117	7.566	7.183	6.916	8.839	7.638
69	4.811	3.965	4.749	0.57	0.743	4.173	1.662	4.728	3.783
70	1.738	2.538	6.182	3.875	3.273	0.36	2.61	5.683	4.454
71	1.733	2.517	6.291	3.949	3.398	0.707	2.719	5.792	4.563
72	1.513	2.867	6.269	4.009	3.458	0.841	2.518	5.77	4.541
73	6.352	4.427	1.379	4.297	4.058	5.748	4.52	0.538	1.375
74	8.53	5.791	3.417	5.995	5.756	7.72	6.218	2.4	3.073
75	3.459	3.261	5.618	2.121	1.752	2.961	1.582	5.119	3.89
76	3.314	2.649	5.1	2.33	1.778	2.732	1.064	4.601	3.372
77	3.23	2.276	5.281	1.965	1.414	2.42	1.396	4.925	3.696
78	2.927	1.973	4.512	2.124	1.573	2.117	1.474	4.036	2.807

Table B.1 continued from previous page

79	4.848	2.774	2.797	2.01	1.772	4.297	2.282	3.06	1.831
80	3.357	2.403	5.298	2.092	1.541	2.547	1.27	4.799	3.57
81	0	3.957	7.256	4.949	4.397	2.002	3.505	6.757	5.528
82	3.957	0	5.198	3.931	3.38	2.898	2.73	4.569	3.508
83	7.256	5.198	0	4.721	4.482	6.446	4.944	1.017	1.799
84	4.949	3.931	4.721	0	0.732	4.139	1.628	4.768	3.749
85	4.397	3.38	4.482	0.732	0	3.587	1.077	4.514	3.511
86	2.002	2.898	6.446	4.139	3.587	0	2.874	5.947	4.718
87	3.505	2.73	4.944	1.628	1.077	2.874	0	4.434	3.205
88	6.757	4.569	1.017	4.768	4.514	5.947	4.434	0	1.778
89	5.528	3.508	1.799	3.749	3.511	4.718	3.205	1.778	0
90	2.875	2.868	5.309	2.519	1.968	2.406	1.281	4.81	3.581
91	3.321	2.61	5.09	2.299	1.748	2.754	1.062	4.591	3.362
92	2.342	1.816	5.405	2.995	2.444	1.186	1.692	4.906	3.677
93	3.766	0.834	5.47	4.199	3.648	2.853	2.998	4.987	3.758
94	4.316	4.426	6.753	3.003	2.644	3.887	2.841	6.378	5.149
95	3.08	4.502	7.583	3.788	3.658	3.765	3.404	7.084	5.855
96	1.242	4.131	7.41	4.209	3.84	2.176	3.448	6.84	5.611
97	4.859	1.055	6.111	4.878	4.327	3.944	3.677	5.6	4.399
98	7.242	5.222	0.356	4.953	4.838	6.432	4.919	0.661	1.762
99	0.786	3.473	6.611	4.28	3.729	1.72	2.789	6.041	4.812
100	0.335	3.671	7.111	4.78	4.229	1.716	3.289	6.541	5.312
101	4.734	4.77	6.923	3.177	2.899	4.265	3.147	6.722	5.493
102	3.898	3.934	6.087	2.341	2.102	3.429	2.311	5.886	4.657
103	4.176	4.212	6.365	2.651	2.294	3.707	2.589	6.164	4.935
104	5.028	5.064	7.217	3.471	3.146	4.559	3.441	7.016	5.787
105	2.894	1.657	6.183	3.773	3.222	2.14	2.48	5.684	4.455
106	2.665	1.654	4.944	2.534	1.983	1.643	1.296	4.445	3.216
107	2.306	1.944	5.718	3.439	2.888	1.28	2.146	5.219	3.99

Table B.1 continued from previous page

108	1.955	2.295	6.069	3.79	3.239	0.929	2.497	5.57	4.341
109	2.214	2.036	5.81	3.531	2.98	1.188	2.238	5.311	4.082
110	5.417	3.397	2.441	3.638	3.4	4.607	3.094	2.42	0.856
111	6.749	2.945	8.001	6.768	6.217	5.834	5.567	7.49	6.289
112	4.953	4.341	5.125	0.946	1.118	4.524	2.028	5.104	3.54
113	2.203	3.778	7.225	4.894	4.343	1.823	3.403	6.655	5.426
114	5.687	7.919	11.365	9.034	8.483	5.964	7.543	10.795	9.566
115	4.666	1.283	6.339	5.106	4.555	3.912	3.905	5.828	4.627
116	4.983	5.059	6.082	1.903	2.075	4.554	2.842	6.061	4.497
117	3.901	3.953	5.506	1.766	1.397	3.472	1.64	5.352	3.921
118	5.122	1.272	5.4	4.918	4.367	4.161	3.502	4.746	3.722
119	4.262	1.741	6.385	5.106	4.555	3.508	3.848	5.902	4.673
120	7.264	4.513	9.149	7.878	7.327	6.51	6.677	8.666	7.437
121	11.516	8.828	13.464	12.192	11.641	10.49	10.991	12.981	11.752
122	5.928	1.971	5.956	5.39	4.839	4.869	4.284	5.302	4.278
123	3.62	1.262	4.295	2.669	2.118	3.024	1.533	3.796	2.567
124	3.377	2.82	5.271	2.453	1.949	2.903	1.235	4.772	3.543
125	4.881	2.175	6.811	5.734	5.182	3.855	4.398	6.328	5.099
126	5.842	2.281	2.942	3.765	3.527	5.159	3.966	2.288	1.357
127	3.381	2.606	4.931	1.615	1.064	2.75	0.124	4.432	3.203
128	4.085	3.395	4.949	1.112	1.017	3.602	1.082	4.795	3.364
129	0.741	3.216	6.586	4.255	3.704	1.261	2.764	6.016	4.787
NODES	90	91	92	93	94	95	96	97	98
depot	4.982	5.343	3.689	5.231	5.808	5.572	3.734	6.446	8.665
1	1.373	1.154	0.626	2.069	2.941	3.202	2.862	2.869	4.864
2	1.38	1.161	1.856	3.06	2.518	3.279	3.547	3.739	4.861
3	1.656	2.051	2.592	3.95	1.628	2.642	2.824	4.629	5.571
4	2.969	2.75	3.445	4.649	3.38	4.053	4.604	5.328	5.329

Table B.1 continued from previous page

5	1.283	2.004	1.58	3.047	2.724	2.284	1.708	3.823	5.859
6	8.145	7.891	7.466	6.484	9.741	10.152	9.781	5.139	9.672
7	5.584	6.067	4.945	6.369	7.065	6.998	5.16	7.462	9.922
8	8.934	8.664	8.255	7.273	10.53	10.941	10.57	5.927	10.461
9	2.155	1.936	2.631	3.835	2.564	3.35	3.788	4.514	4.514
10	2.458	2.239	2.934	4.138	2.765	3.727	3.989	4.817	4.818
11	2.514	2.295	2.99	4.194	3.008	3.783	4.273	4.873	4.583
12	0.885	0.666	1.141	2.609	2.481	3.187	2.804	3.353	4.977
13	4.273	4.153	3.098	1.824	5.754	6.017	5.033	2.563	7.085
14	4.228	4.155	3.052	1.826	5.709	5.961	4.987	2.506	6.689
15	2.459	2.24	2.433	1.663	4.055	4.761	4.327	2.304	3.865
16	3.391	3.27	2.216	0.942	4.872	5.135	4.324	0.709	5.864
17	3.1	2.881	3.576	4.78	3.511	4.184	4.735	5.459	5.502
18	2.75	2.531	3.226	4.43	3.161	3.834	4.385	5.109	5.183
19	2.819	3.195	3.755	5.143	2.791	3.756	3.923	5.875	6.031
20	2.782	2.525	1.729	0.748	4.365	4.628	4.044	1.299	5.457
21	1.866	1.647	2.342	3.546	2.629	3.501	3.825	4.225	4.486
22	3.039	2.82	3.516	4.719	3.466	4.308	4.69	5.398	5.399
23	3.114	2.895	3.21	3.308	4.472	5.344	5.145	3.949	3.029
24	1.569	1.925	2.505	3.842	1.299	2.555	2.638	4.579	6.009
25	5.705	5.448	4.652	3.541	7.288	7.253	6.923	2.146	8.003
26	3.371	3.011	2.216	1.234	4.852	5.01	4.532	0.547	5.566
27	1.31	1.115	0.692	2.16	2.791	2.958	2.355	2.935	4.971
28	0.996	1.479	1.055	2.491	2.477	2.74	2.082	3.298	5.334
29	1.681	1.733	0.505	1.712	3.162	3.425	2.228	2.805	5.541
30	1.575	1.627	0.399	1.606	3.056	3.319	2.334	2.699	5.435
31	5.049	4.83	5.145	4.78	6.436	7.308	7.092	5.001	3.109
32	0.679	1.541	1.558	2.994	1.97	2.091	1.488	3.801	5.837
33	4.446	4.07	3.847	2.865	6.014	6.547	6.162	2.347	6.174

Table B.1 continued from previous page

34	4.499	4.123	3.698	2.716	6.067	6.384	6.013	2.208	6.103
35	1.364	1.697	2.33	3.58	1.082	2.497	2.391	4.399	5.781
36	2.201	2.684	1.465	2.796	3.682	3.625	1.971	3.983	6.539
37	6.762	6.641	5.587	4.313	8.243	8.507	7.459	5.052	9.235
38	2.611	2.392	2.585	1.815	4.207	4.913	4.479	2.305	4.017
39	2.319	2.1	2.249	1.895	3.887	4.285	3.842	2.536	3.577
40	1.514	2.235	1.812	3.246	2.955	2.542	1.477	4.054	6.09
41	2.877	2.929	1.701	0.658	4.358	4.621	3.636	1.397	5.918
42	11.575	11.321	10.896	9.914	13.171	13.582	13.211	8.568	13.102
43	4.374	3.984	3.762	2.78	5.942	6.462	6.077	2.262	6.089
44	1.732	1.784	0.556	1.449	3.213	3.476	2.491	2.542	5.592
45	1.315	1.096	1.41	2.086	2.883	3.589	3.444	2.765	3.981
46	1.683	1.464	1.779	2.199	3.251	3.957	3.806	2.878	3.612
47	4.53	4.311	4.626	4.714	5.997	6.804	6.653	5.355	0.92
48	3.329	3.11	3.28	2.298	4.925	5.58	5.137	1.857	4.872
49	1.849	2.457	2.506	3.787	2.514	1.226	1.375	4.601	6.763
50	1.402	2.264	2.281	3.672	2.616	1.744	0.857	4.486	6.56
51	6.826	6.607	6.922	6.614	8.27	9.1	8.926	6.835	4.943
52	1.356	1.805	2.292	3.753	1.328	2.342	2.524	4.507	5.85
53	1.332	1.113	1.165	2.101	2.928	3.634	3.401	2.591	4.623
54	3.79	3.414	3.191	2.209	5.358	5.891	5.506	1.691	5.518
55	2.277	2.058	2.787	3.938	3.687	4.489	4.444	4.579	3.957
56	1.998	1.779	2.093	2.765	3.566	4.272	4.127	3.444	4.275
57	1.888	1.669	1.952	2.173	3.456	4.162	4.018	2.814	3.709
58	3.582	3.462	2.407	1.133	5.063	5.326	4.353	1.872	6.394
59	2.752	2.533	3.262	3.463	3.632	4.504	4.828	4.104	3.482
60	5.823	6.306	5.113	6.609	7.232	6.996	5.158	7.702	10.089
61	1.325	1.106	1.125	2.371	2.893	3.512	3.159	2.926	4.266
62	0.701	0.482	1.177	2.612	2.297	3.003	2.868	3.306	4.594

Table B.1 continued from previous page

63	1.238	1.369	0.946	2.333	2.719	2.982	2.539	3.07	4.88
64	1.086	0.867	0.994	2.448	2.682	3.276	2.673	3.152	4.776
65	9.796	9.526	9.117	8.135	11.392	11.803	11.432	6.789	11.323
66	2.165	1.962	1.199	1.047	3.646	3.697	3.466	2.148	5.656
67	7.259	6.99	6.331	5.22	8.855	8.932	8.602	3.825	8.802
68	7.054	6.797	6.001	4.89	8.637	8.602	8.272	3.495	9.352
69	2.553	2.334	3.029	4.233	2.766	3.731	3.962	4.912	4.913
70	2.142	2.491	0.922	2.589	3.623	3.501	1.912	3.593	6.168
71	2.137	2.6	1.047	2.146	3.618	3.496	1.907	3.376	6.277
72	1.917	2.4	1.181	2.496	3.398	3.276	1.687	3.726	6.255
73	4.865	4.666	4.658	5.046	5.973	6.679	6.559	5.482	1.023
74	6.583	6.364	6.667	6.601	7.982	8.688	8.684	6.822	3.061
75	1.079	1.664	2.308	3.579	2.439	2.803	3.016	4.258	5.604
76	0.669	1.344	1.669	3.104	1.777	2.483	2.606	3.704	5.086
77	1.057	0.334	1.993	3.11	3.113	3.391	2.947	3.331	5.41
78	1.679	1.444	1.027	2.371	3.247	3.414	3.297	3.028	4.521
79	3.173	2.954	3.269	3.608	4.048	4.92	4.932	3.829	3.153
80	0.931	0.208	1.867	3.217	2.987	3.265	2.821	3.458	5.284
81	2.875	3.321	2.342	3.766	4.316	3.08	1.242	4.859	7.242
82	2.868	2.61	1.816	0.834	4.426	4.502	4.131	1.055	5.222
83	5.309	5.09	5.405	5.47	6.753	7.583	7.41	6.111	0.356
84	2.519	2.299	2.995	4.199	3.003	3.788	4.209	4.878	4.953
85	1.968	1.748	2.444	3.648	2.644	3.658	3.84	4.327	4.838
86	2.406	2.754	1.186	2.853	3.887	3.765	2.176	3.944	6.432
87	1.281	1.062	1.692	2.998	2.841	3.404	3.448	3.677	4.919
88	4.81	4.591	4.906	4.987	6.378	7.084	6.84	5.6	0.661
89	3.581	3.362	3.677	3.758	5.149	5.855	5.611	4.399	1.762
90	0	1.139	1.745	3.181	2.338	2.611	2.167	3.918	5.295
91	1.139	0	1.659	3.009	2.779	3.057	2.614	3.558	5.076

Table B.1 continued from previous page

92	1.745	1.659	0	1.824	3.003	3.266	2.516	2.762	5.391
93	3.181	3.009	1.824	0	4.48	4.743	3.94	1.651	5.472
94	2.338	2.779	3.003	4.48	0	3.3	3.458	5.399	6.863
95	2.611	3.057	3.266	4.743	3.3	0	1.838	5.363	7.426
96	2.167	2.614	2.516	3.94	3.458	1.838	0	5.033	7.325
97	3.918	3.558	2.762	1.651	5.399	5.363	5.033	0	6.113
98	5.295	5.076	5.391	5.472	6.863	7.426	7.325	6.113	0
99	2.089	2.535	2.06	3.484	3.53	2.739	0.901	4.419	6.526
100	2.589	3.035	2.056	3.48	4.03	2.794	0.956	4.573	7.026
101	2.682	3.123	3.381	4.858	1.719	3.285	3.851	5.776	7.207
102	1.846	2.287	2.545	4.022	0.851	2.449	3.086	4.94	6.371
103	2.124	2.565	2.823	4.3	1.161	2.727	3.364	5.218	6.649
104	2.976	3.417	3.675	5.152	2.013	3.579	4.145	6.07	7.501
105	2.309	2.361	1.133	1.374	3.79	3.908	3.068	2.113	6.169
106	1.417	1.198	0.461	1.896	2.951	2.848	2.771	2.6	4.93
107	1.975	2.027	0.799	1.732	3.456	3.574	2.48	2.825	5.704
108	2.322	2.378	1.15	1.924	3.803	3.718	2.129	3.154	6.055
109	2.067	2.119	0.891	1.665	3.548	3.666	2.388	2.895	5.796
110	3.47	3.251	3.566	3.647	5.038	5.601	5.5	4.288	2.605
111	5.705	5.448	4.652	3.541	7.288	7.253	6.923	2.146	8.003
112	2.929	2.71	3.405	4.609	2.871	3.544	4.095	5.288	5.289
113	2.802	3.285	2.163	3.587	4.283	4.215	2.377	4.68	7.14
114	6.942	7.425	6.232	7.728	8.351	8.115	6.277	8.821	11.208
115	3.908	3.786	2.733	1.458	5.389	5.591	4.84	0.56	6.341
116	2.971	3.412	3.67	5.147	2.901	3.574	4.125	6.065	6.246
117	1.889	2.322	2.588	4.065	1.819	2.492	3.043	4.9	5.67
118	3.959	3.74	2.98	1.998	5.476	5.697	5.296	1.227	5.407
119	3.677	3.729	2.501	1.458	5.158	5.421	4.436	2.197	6.387
120	6.679	6.688	5.503	4.229	8.16	8.423	7.438	4.968	9.151

Table B.1 continued from previous page

121	10.993	10.872	9.818	8.544	12.474	12.738	11.69	9.283	13.466
122	4.586	4.212	3.787	2.805	6.156	6.388	6.102	2.286	5.963
123	1.87	1.651	1.934	1.53	3.438	3.936	3.794	2.209	4.281
124	0.732	1.173	1.84	3.088	1.948	2.312	2.669	3.767	5.257
125	4.352	4.35	3.165	1.891	5.833	5.929	5.055	2.63	6.813
126	4.135	3.916	4.096	3.115	5.731	6.013	5.925	3.336	2.949
127	1.405	1.186	1.568	2.874	2.717	3.28	3.464	3.553	4.917
128	1.983	1.764	2.459	3.663	2.003	2.676	3.227	4.342	5.113
129	2.163	2.646	1.601	3.025	3.644	2.753	0.915	4.118	6.501
NODES	99	100	101	102	103	104	105	106	107
depot	3.278	3.274	6.186	5.35	5.628	6.48	4.518	4.145	3.658
1	2.063	2.554	3.285	2.449	2.727	3.579	1.414	0.354	1.08
2	3.141	3.641	2.773	1.944	2.168	3.02	2.634	1.395	2.3
3	2.896	3.396	1.883	1.086	1.278	2.13	3.218	2.285	2.884
4	4.676	5.176	3.554	2.718	2.996	3.848	4.223	2.984	3.889
5	1.124	1.624	3.142	2.306	2.584	3.436	2.214	1.308	1.88
6	9.123	9.321	10.117	9.249	9.527	10.381	6.996	7.304	7.594
7	4.704	4.999	7.49	6.693	6.885	7.739	5.497	5.268	4.909
8	9.912	10.11	10.906	10.038	10.316	11.17	7.784	8.093	8.383
9	3.844	4.344	2.738	1.902	2.212	3.032	3.38	2.17	3.046
10	4.061	4.561	2.939	2.103	2.413	3.233	3.683	2.473	3.378
11	4.275	4.775	3.264	2.448	2.659	3.511	3.768	2.529	3.434
12	2.005	2.505	2.872	2.217	2.267	3.121	1.774	0.753	1.44
13	4.577	4.573	6.179	5.382	5.574	6.428	2.124	3.17	2.825
14	4.531	4.527	6.173	5.337	5.576	6.43	2.067	3.113	2.779
15	3.528	4.028	4.446	3.766	3.841	4.695	2.494	1.972	2.773
16	3.868	3.864	5.297	4.5	4.692	5.546	1.404	2.288	2.116
17	4.807	5.307	3.685	2.849	3.127	3.979	4.354	3.115	4.02

Table B.1 continued from previous page

18	4.457	4.957	3.335	2.499	2.777	3.629	4.004	2.765	3.67
19	4.059	4.559	2.858	2.022	2.332	3.152	4.381	3.401	4.047
20	3.386	3.584	4.722	3.886	4.196	5.016	1.719	1.842	1.858
21	3.627	4.127	2.793	1.957	2.235	3.087	3.12	1.881	2.786
22	4.753	5.253	3.64	2.804	3.114	3.934	4.264	3.055	3.93
23	4.345	4.845	4.636	3.8	4.078	4.93	3.988	2.749	3.523
24	2.71	3.21	1.284	0.448	0.726	1.578	2.97	2.131	2.636
25	6.309	6.463	7.645	6.809	7.119	7.939	4.003	4.599	4.715
26	3.873	4.077	5.278	4.442	4.72	5.572	2.109	2.162	2.344
27	1.556	2.056	3.012	2.176	2.454	3.306	1.325	0.486	0.991
28	1.192	1.692	2.941	2.105	2.383	3.235	1.619	0.78	1.285
29	1.772	1.768	3.626	2.79	3.068	3.92	0.84	0.611	0.506
30	1.878	1.874	3.52	2.684	2.962	3.814	0.734	0.505	0.4
31	6.293	6.793	6.802	5.973	6.197	7.049	5.627	4.684	5.458
32	1.56	2.06	2.434	1.598	1.876	2.728	2.122	1.283	1.788
33	5.504	5.702	6.358	5.522	5.8	6.652	3.688	3.699	3.975
34	5.355	5.553	6.411	5.575	5.853	6.705	3.539	3.536	3.826
35	2.489	2.989	1.616	0.78	1.058	1.91	2.708	1.869	2.374
36	1.515	1.511	4.125	3.289	3.567	4.419	2.018	1.789	1.223
37	7.003	6.999	8.707	7.871	8.149	9.001	4.602	5.659	5.163
38	3.68	4.18	4.598	3.839	3.993	4.847	2.646	2.124	2.925
39	3.043	3.543	4.231	3.395	3.673	4.525	2.764	1.922	2.43
40	0.576	1.076	3.419	2.583	2.861	3.713	2.374	1.535	2.062
41	3.18	3.176	4.822	3.986	4.264	5.116	0.716	1.807	1.428
42	12.553	12.751	13.547	12.679	12.957	13.811	10.425	10.734	11.024
43	5.419	5.617	6.286	5.45	5.728	6.58	3.603	3.614	3.903
44	2.035	2.031	3.677	2.841	3.119	3.971	0.577	0.662	0.283
45	2.645	3.145	3.227	2.391	2.669	3.521	2.188	0.949	1.821
46	3.014	3.514	3.595	2.759	3.037	3.889	2.557	1.318	2.092

Table B.1 continued from previous page

47	5.855	6.355	6.332	5.496	5.806	6.626	5.404	4.165	4.962
48	4.338	4.838	5.269	4.562	4.84	5.563	3.121	2.991	3.408
49	2.182	2.331	2.769	1.901	2.179	3.031	2.884	2.045	2.843
50	1.749	1.813	2.994	2.244	2.522	3.288	2.769	1.93	2.511
51	8.127	8.627	8.605	7.769	8.079	8.899	7.437	6.461	7.235
52	2.596	3.096	1.515	0.718	0.91	1.762	2.85	2.011	2.516
53	2.602	3.102	3.272	2.436	2.746	3.566	2.04	0.813	1.479
54	4.848	5.046	5.702	4.866	5.144	5.996	3.032	3.043	3.319
55	4.038	4.538	3.83	3.033	3.225	4.077	3.502	2.502	3.168
56	3.328	3.828	3.91	3.074	3.352	4.204	2.871	1.632	2.504
57	3.219	3.719	3.8	2.964	3.242	4.094	2.73	1.491	2.396
58	3.897	3.893	5.441	4.605	4.883	5.735	1.433	2.479	2.145
59	4.513	5.013	3.797	2.961	3.239	4.091	3.977	2.885	3.643
60	4.702	4.698	7.61	6.774	7.052	7.904	5.737	5.508	5.082
61	2.36	2.86	3.237	2.401	2.679	3.531	1.903	0.664	1.569
62	2.648	3.095	2.641	1.805	2.083	2.935	1.955	0.716	1.621
63	1.74	2.24	3.097	2.261	2.539	3.391	1.461	0.674	1.127
64	1.874	2.374	3.073	2.418	2.468	3.322	1.643	0.552	1.309
65	10.774	10.972	11.736	10.9	11.178	12.03	8.646	8.955	9.245
66	2.667	3.054	4.024	3.188	3.466	4.318	1.463	0.849	1.489
67	7.988	8.142	9.199	8.363	8.641	9.493	5.682	6.278	6.394
68	7.658	7.812	8.994	8.158	8.457	9.288	5.352	5.839	6.064
69	4.034	4.534	3.021	2.198	2.416	3.268	3.778	2.568	3.473
70	1.456	1.452	4.001	3.165	3.443	4.295	1.876	1.379	1.016
71	1.451	1.447	3.996	3.16	3.438	4.29	1.433	1.478	0.573
72	1.231	1.227	3.776	2.94	3.218	4.07	1.783	1.637	0.923
73	5.76	6.26	6.317	5.481	5.759	6.611	5.436	4.197	5.102
74	7.885	8.385	8.197	7.361	7.639	8.491	7.424	6.206	6.992
75	2.938	3.438	2.669	1.833	2.111	2.963	3.096	1.87	2.53

Table B.1 continued from previous page

76	2.528	3.028	2.121	1.285	1.563	2.415	2.447	1.208	2.113
77	2.869	3.369	3.457	2.621	2.899	3.751	2.695	1.532	2.361
78	2.498	2.945	3.591	2.755	3.033	3.885	1.805	0.566	1.471
79	4.133	4.633	4.218	3.382	3.66	4.512	3.902	3.063	3.568
80	2.743	3.243	3.331	2.495	2.773	3.625	2.569	1.406	2.235
81	0.786	0.335	4.734	3.898	4.176	5.028	2.894	2.665	2.306
82	3.473	3.671	4.77	3.934	4.212	5.064	1.657	1.654	1.944
83	6.611	7.111	6.923	6.087	6.365	7.217	6.183	4.944	5.718
84	4.28	4.78	3.177	2.341	2.651	3.471	3.773	2.534	3.439
85	3.729	4.229	2.899	2.102	2.294	3.146	3.222	1.983	2.888
86	1.72	1.716	4.265	3.429	3.707	4.559	2.14	1.643	1.28
87	2.789	3.289	3.147	2.311	2.589	3.441	2.48	1.296	2.146
88	6.041	6.541	6.722	5.886	6.164	7.016	5.684	4.445	5.219
89	4.812	5.312	5.493	4.657	4.935	5.787	4.455	3.216	3.99
90	2.089	2.589	2.682	1.846	2.124	2.976	2.309	1.417	1.975
91	2.535	3.035	3.123	2.287	2.565	3.417	2.361	1.198	2.027
92	2.06	2.056	3.381	2.545	2.823	3.675	1.133	0.461	0.799
93	3.484	3.48	4.858	4.022	4.3	5.152	1.374	1.896	1.732
94	3.53	4.03	1.719	0.851	1.161	2.013	3.79	2.951	3.456
95	2.739	2.794	3.285	2.449	2.727	3.579	3.908	2.848	3.574
96	0.901	0.956	3.851	3.086	3.364	4.145	3.068	2.771	2.48
97	4.419	4.573	5.776	4.94	5.218	6.07	2.113	2.6	2.825
98	6.526	7.026	7.207	6.371	6.649	7.501	6.169	4.93	5.704
99	0	0.5	3.948	3.112	3.39	4.242	2.612	1.972	2.024
100	0.5	0	4.448	3.612	3.89	4.742	2.608	2.379	2.02
101	3.948	4.448	0	0.868	0.605	0.294	4.254	3.329	3.92
102	3.112	3.612	0.868	0	0.31	1.162	3.418	2.493	3.084
103	3.39	3.89	0.605	0.31	0	0.887	3.696	2.771	3.362
104	4.242	4.742	0.294	1.162	0.887	0	4.548	3.623	4.214

Table B.1 continued from previous page

105	2.612	2.608	4.254	3.418	3.696	4.548	0	1.239	0.86
106	1.972	2.379	3.329	2.493	2.771	3.623	1.239	0	0.905
107	2.024	2.02	3.92	3.084	3.362	4.214	0.86	0.905	0
108	1.673	1.669	4.218	3.382	3.66	4.512	1.211	1.256	0.351
109	1.932	1.928	4.012	3.176	3.454	4.306	0.952	0.997	0.092
110	4.701	5.201	5.32	4.484	4.762	5.614	4.344	3.105	3.879
111	6.309	6.463	7.645	6.809	7.119	7.939	4.003	4.599	4.715
112	4.167	4.667	3.045	2.209	2.487	3.339	4.183	2.944	3.849
113	1.921	2.216	4.747	3.911	4.189	5.041	2.715	2.486	2.127
114	5.821	5.817	8.729	7.893	8.171	9.023	6.856	6.627	6.201
115	4.384	4.38	5.814	5.017	5.209	6.063	1.92	2.805	2.632
116	4.197	4.697	3.075	2.239	2.517	3.369	4.457	3.618	4.123
117	3.115	3.615	1.993	1.157	1.435	2.287	3.375	2.536	3.041
118	4.637	4.841	5.706	4.87	5.148	6	2.873	2.926	3.108
119	3.98	3.976	5.622	4.786	5.064	5.916	1.516	2.607	2.228
120	6.982	6.978	8.624	7.788	8.066	8.918	4.518	5.575	5.23
121	11.234	11.23	12.938	12.102	12.38	13.232	8.833	9.89	9.394
122	5.444	5.642	6.5	5.664	5.942	6.794	3.628	3.625	3.915
123	3.114	3.614	3.782	2.946	3.224	4.076	2.353	1.473	2.377
124	2.591	3.091	2.178	1.342	1.62	2.472	2.618	1.379	2.284
125	4.599	4.595	6.297	5.461	5.739	6.591	2.191	3.237	2.759
126	5.126	5.626	5.967	5.131	5.409	6.261	3.938	3.648	4.225
127	2.665	3.165	3.048	2.212	2.49	3.342	2.356	1.356	2.022
128	3.299	3.799	2.177	1.341	1.619	2.471	3.237	1.998	2.903
129	0.459	0.754	4.108	3.272	3.55	4.402	2.153	1.924	1.565
NODES	108	109	110	111	112	113	114	115	116
depot	3.307	3.566	6.84	8.336	6.843	3.881	2.543	6.29	6.475
1	1.431	1.172	3.039	4.759	2.9	2.661	6.802	2.978	3.574

Table B.1 continued from previous page

2	2.651	2.392	3.067	5.629	1.549	3.755	7.895	3.967	2.353
3	3.187	2.976	3.639	6.519	1.433	3.667	7.807	4.773	1.463
4	4.24	3.981	3.828	7.218	0.509	5.344	9.484	5.556	1.467
5	1.714	1.972	4.034	5.712	3.361	2.193	6.334	3.743	3.391
6	7.945	7.686	7.847	2.993	9.475	9.428	13.569	5.443	10.283
7	4.558	4.817	8.097	9.352	7.702	2.81	5.347	7.269	7.732
8	8.734	8.475	8.636	3.781	10.248	10.217	14.358	6.231	11.056
9	3.397	3.138	3.365	6.404	0.848	4.483	8.623	4.742	1.805
10	3.729	3.47	3.577	6.707	0.592	4.786	8.926	5.045	1.508
11	3.785	3.526	3.434	6.763	0.941	4.889	9.029	5.101	1.898
12	1.791	1.532	3.152	5.27	2.459	2.619	6.759	3.373	3.114
13	3.176	2.917	5.26	4.298	5.883	4.68	7.97	2.37	6.421
14	3.13	2.871	4.864	2.329	5.885	4.634	8.775	2.314	6.376
15	3.037	2.778	2.04	4.194	3.799	4.142	8.282	2.532	4.751
16	2.467	2.208	4.039	2.599	5	3.971	8.112	0.517	5.539
17	4.371	4.112	3.959	7.349	0.64	5.475	9.615	5.687	1.598
18	4.021	3.762	3.609	6.999	0.29	5.125	9.265	5.337	1.248
19	4.29	4.139	4.061	7.765	0.742	4.913	9.053	5.839	0.217
20	2.209	1.95	3.632	3.189	4.255	3.691	7.832	1.527	5.011
21	3.137	2.878	3.047	6.115	1.604	4.241	8.381	4.453	2.561
22	4.281	4.022	4.137	7.288	0.818	5.367	9.507	5.626	1.776
23	3.874	3.615	1.204	5.839	2.936	4.959	9.099	4.177	3.803
24	2.983	2.728	4.122	6.468	1.847	3.463	7.603	4.569	1.877
25	5.066	4.807	6.178	0	7.178	6.57	10.711	2.45	7.934
26	2.695	2.436	3.741	2.437	4.741	4.179	8.32	0.775	5.519
27	1.342	1.083	3.146	4.823	2.908	2.17	6.31	2.924	3.281
28	1.326	1.377	3.509	5.117	3.114	1.806	5.946	3.218	3.144
29	0.857	0.598	3.716	4.695	3.555	1.875	6.016	2.612	3.829
30	0.751	0.492	3.61	4.589	3.449	1.981	6.122	2.506	3.723

Table B.1 continued from previous page

31	5.809	5.55	2.124	6.891	4.808	6.907	11.047	5.229	5.765
32	2.139	1.88	4.012	5.62	2.607	2.629	6.77	3.791	2.637
33	4.326	4.067	4.349	4.237	5.654	5.809	9.95	2.576	6.462
34	4.177	3.918	4.278	4.098	5.707	5.66	9.801	2.437	6.515
35	2.725	2.466	3.956	6.206	1.789	3.65	7.791	4.307	1.819
36	0.872	1.131	4.714	5.873	4.319	1.618	5.759	3.79	4.349
37	5.33	5.071	7.41	6.595	8.371	5.487	8.542	4.859	8.91
38	3.189	2.93	2.192	4.195	3.951	4.294	8.434	2.533	4.903
39	2.781	2.522	1.752	4.426	3.511	3.657	7.797	2.764	4.332
40	1.711	1.97	4.265	5.872	3.592	1.959	5.931	3.973	3.622
41	1.779	1.52	4.093	3.287	4.716	3.283	7.424	1.204	5.025
42	11.375	11.116	11.277	6.422	12.905	12.858	16.999	8.872	13.713
43	4.254	3.995	4.264	4.152	5.569	5.724	9.865	2.491	6.39
44	0.634	0.375	3.767	4.432	3.606	2.138	6.279	2.349	3.88
45	2.172	1.913	2.156	4.655	2.527	3.259	7.399	2.993	3.331
46	2.443	2.184	1.787	4.768	2.83	3.628	7.768	3.106	3.634
47	5.313	5.054	1.685	7.245	4.369	6.469	10.609	5.583	5.326
48	3.759	3.5	3.047	3.747	4.698	4.952	9.092	2.086	5.502
49	2.492	2.751	4.938	6.382	3.059	3.251	7.392	4.696	3.006
50	2.376	2.603	4.735	6.267	3.253	2.818	6.959	4.514	3.283
51	7.586	7.327	3.958	8.725	6.642	8.741	12.881	7.063	7.599
52	2.867	2.608	4.007	6.327	1.801	3.533	7.674	4.525	1.748
53	1.83	1.571	2.798	4.373	2.906	3.215	7.356	2.819	3.561
54	3.67	3.411	3.693	3.581	4.998	5.153	9.294	1.92	5.806
55	3.519	3.26	2.616	6.469	2.48	4.652	8.792	4.807	3.437
56	2.855	2.596	2.45	5.334	2.85	3.942	8.082	3.672	3.654
57	2.747	2.488	1.884	4.704	3.097	3.833	7.973	3.042	3.901
58	2.496	2.237	4.569	3.607	5.192	4	8.141	1.679	5.73
59	3.994	3.735	2.141	5.994	2.005	5.127	9.267	4.332	2.962

Table B.1 continued from previous page

60	4.731	4.99	8.264	9.592	7.941	4.689	1.119	7.509	7.899
61	1.92	1.661	2.441	4.816	2.812	2.974	7.114	3.154	3.526
62	1.972	1.713	2.769	5.088	2.228	3.202	7.343	3.521	2.93
63	1.478	1.219	3.055	4.96	3.162	2.354	6.494	3.178	3.386
64	1.66	1.401	2.951	5.069	2.66	2.488	6.628	3.242	3.315
65	9.596	9.337	9.498	4.643	11.11	11.079	15.22	7.093	11.918
66	1.84	1.581	3.831	4.038	3.755	3.161	7.302	1.956	4.313
67	6.723	6.464	6.977	1.679	8.574	8.249	12.39	4.129	9.382
68	6.415	6.156	7.527	1.349	8.527	7.919	12.06	3.799	9.283
69	3.824	3.565	3.672	6.802	0.687	4.881	9.021	5.14	1.413
70	0.665	0.924	4.343	5.483	4.26	1.559	5.7	3.648	4.29
71	0.222	0.481	4.452	5.288	4.255	1.554	5.695	3.205	4.285
72	0.572	0.831	4.43	5.638	4.035	1.334	5.475	3.555	4.065
73	5.453	5.194	2.017	7.372	4.701	6.374	10.514	5.71	5.658
74	7.343	7.084	3.715	8.712	6.399	8.499	12.639	7.05	7.356
75	2.881	2.622	3.779	6.098	2.87	3.795	7.863	4.486	2.937
76	2.464	2.205	3.261	5.58	2.38	3.425	7.543	3.932	2.41
77	2.712	2.453	3.585	5.221	2.532	3.619	7.759	3.559	3.489
78	1.822	1.563	2.696	4.918	2.534	3.052	7.193	3.256	3.338
79	3.919	3.66	1.72	5.719	2.42	4.747	8.887	4.057	3.377
80	2.586	2.327	3.459	5.348	2.659	3.493	7.633	3.686	3.616
81	1.955	2.214	5.417	6.749	4.953	2.203	5.687	4.666	4.983
82	2.295	2.036	3.397	2.945	4.341	3.778	7.919	1.283	5.059
83	6.069	5.81	2.441	8.001	5.125	7.225	11.365	6.339	6.082
84	3.79	3.531	3.638	6.768	0.946	4.894	9.034	5.106	1.903
85	3.239	2.98	3.4	6.217	1.118	4.343	8.483	4.555	2.075
86	0.929	1.188	4.607	5.834	4.524	1.823	5.964	3.912	4.554
87	2.497	2.238	3.094	5.567	2.028	3.403	7.543	3.905	2.842
88	5.57	5.311	2.42	7.49	5.104	6.655	10.795	5.828	6.061

Table B.1 continued from previous page

89	4.341	4.082	0.856	6.289	3.54	5.426	9.566	4.627	4.497
90	2.322	2.067	3.47	5.705	2.929	2.802	6.942	3.908	2.971
91	2.378	2.119	3.251	5.448	2.71	3.285	7.425	3.786	3.412
92	1.15	0.891	3.566	4.652	3.405	2.163	6.232	2.733	3.67
93	1.924	1.665	3.647	3.541	4.609	3.587	7.728	1.458	5.147
94	3.803	3.548	5.038	7.288	2.871	4.283	8.351	5.389	2.901
95	3.718	3.666	5.601	7.253	3.544	4.215	8.115	5.591	3.574
96	2.129	2.388	5.5	6.923	4.095	2.377	6.277	4.84	4.125
97	3.154	2.895	4.288	2.146	5.288	4.68	8.821	0.56	6.065
98	6.055	5.796	2.605	8.003	5.289	7.14	11.208	6.341	6.246
99	1.673	1.932	4.701	6.309	4.167	1.921	5.821	4.384	4.197
100	1.669	1.928	5.201	6.463	4.667	2.216	5.817	4.38	4.697
101	4.218	4.012	5.32	7.645	3.045	4.747	8.729	5.814	3.075
102	3.382	3.176	4.484	6.809	2.209	3.911	7.893	5.017	2.239
103	3.66	3.454	4.762	7.119	2.487	4.189	8.171	5.209	2.517
104	4.512	4.306	5.614	7.939	3.339	5.041	9.023	6.063	3.369
105	1.211	0.952	4.344	4.003	4.183	2.715	6.856	1.92	4.457
106	1.256	0.997	3.105	4.599	2.944	2.486	6.627	2.805	3.618
107	0.351	0.092	3.879	4.715	3.849	2.127	6.201	2.632	4.123
108	0	0.259	4.23	5.066	4.2	1.776	5.85	2.983	4.47
109	0.259	0	3.971	4.807	3.941	2.035	6.109	2.724	4.215
110	4.23	3.971	0	6.178	3.319	5.315	9.383	4.516	4.276
111	5.066	4.807	6.178	0	7.178	6.57	10.711	2.45	7.934
112	4.2	3.941	3.319	7.178	0	4.92	9.06	5.516	0.958
113	1.776	2.035	5.315	6.57	4.92	0	5.351	4.487	4.95
114	5.85	6.109	9.383	10.711	9.06	5.351	0	8.628	9.018
115	2.983	2.724	4.516	2.45	5.516	4.487	8.628	0	6.056
116	4.47	4.215	4.276	7.934	0.958	4.95	9.018	6.056	0
117	3.388	3.133	3.7	6.79	1.45	3.868	7.936	4.974	1.48

Table B.1 continued from previous page

118	3.459	3.2	3.611	3.117	5.328	4.943	9.084	1.456	5.901
119	2.579	2.32	4.562	4.087	5.516	4.083	8.224	2.004	5.825
120	5.581	5.322	7.326	3.97	8.288	5.405	8.387	4.776	8.827
121	9.561	9.302	11.641	10.826	12.602	9.718	12.701	9.09	13.141
122	4.266	4.007	4.167	4.176	5.796	5.749	9.89	2.515	6.604
123	2.728	2.469	2.456	4.099	3.079	3.728	7.868	2.437	3.883
124	2.635	2.376	3.432	5.657	2.416	3.304	7.372	3.995	2.446
125	2.926	2.667	4.988	2.685	6.143	4.702	8.776	2.438	6.5
126	4.576	4.317	2.033	5.226	4.267	5.74	9.88	3.564	5.134
127	2.373	2.114	3.092	5.443	2.025	3.279	7.419	3.781	2.829
128	3.254	2.995	3.143	6.232	1.521	4.052	8.12	4.57	1.944
129	1.214	1.473	4.676	6.008	4.281	1.462	5.362	3.925	4.311
NODES	117	118	119	120	121	122	123	124	125
depot	5.393	6.669	5.886	5.844	10.158	7.471	5.618	4.829	6.233
1	2.492	3.086	2.782	5.748	10.063	3.858	1.429	1.335	3.41
2	1.161	3.779	3.967	6.739	11.053	4.251	1.53	1.361	4.594
3	0.381	4.453	4.594	7.596	11.91	5.141	2.42	1.067	5.269
4	1.959	5.368	5.556	8.328	12.642	5.84	3.119	2.902	6.183
5	2.309	4.04	3.582	6.584	10.898	4.777	2.447	1.785	4.257
6	9.087	4.749	7.08	6.962	13.818	3.854	6.773	8.1	5.678
7	6.65	7.73	6.149	5.919	10.233	8.531	6.51	6.086	7.375
8	9.86	5.538	7.868	7.75	14.606	4.643	7.562	8.873	6.466
9	1.328	4.554	4.742	7.514	11.828	5.022	2.305	2.014	5.369
10	1.334	4.857	5.045	7.817	12.131	5.325	2.608	2.3	5.672
11	1.761	4.664	5.101	7.873	12.187	5.22	2.664	2.504	5.766
12	2.032	3.486	3.142	6.144	10.458	4.042	1.565	0.875	3.817
13	5.339	3.324	0.867	2.553	6.868	4.079	2.804	4.362	2.093
14	5.294	3.215	1.808	3.671	8.656	4.022	2.806	4.364	0.356

Table B.1 continued from previous page

15	3.414	1.571	2.578	5.342	9.657	2.127	1.087	2.449	3.004
16	4.457	1.604	1.488	4.259	8.574	2.663	1.921	3.479	1.921
17	2.09	5.499	5.687	8.459	12.773	5.967	3.25	3.033	6.314
18	1.74	5.149	5.337	8.109	12.423	5.621	2.9	2.683	5.964
19	1.264	5.685	5.757	8.757	13.072	6.387	3.666	2.23	6.432
20	3.867	1.517	1.662	4.427	8.742	2.325	1.176	2.734	2.089
21	1.382	4.265	4.453	7.225	11.539	4.737	2.016	1.847	5.08
22	2.035	5.438	5.626	8.398	12.712	5.906	3.189	2.973	6.253
23	3.225	3.338	4.223	6.987	11.302	3.947	2.1	3.076	4.649
24	0.795	4.508	4.338	7.34	11.654	5.302	2.584	0.98	5.013
25	6.79	3.117	4.087	3.97	10.826	4.176	4.099	5.657	2.685
26	4.353	0.764	2.148	5.021	9.228	1.823	1.662	3.22	2.575
27	2.282	3.152	2.693	5.695	10.009	3.958	1.559	1.324	3.368
28	2.062	3.515	2.987	5.989	10.303	4.252	1.922	1.498	3.662
29	2.747	3.073	2.208	5.21	9.525	3.874	2.084	1.99	2.883
30	2.641	2.967	2.102	5.104	9.419	3.768	1.978	1.884	2.777
31	5.189	4.147	5.694	8.459	12.774	4.703	4.035	5.011	6.121
32	1.555	4.018	3.49	6.492	10.806	4.755	2.425	1.181	4.165
33	5.266	1.12	3.772	6.544	10.859	1.067	3.293	4.279	4.206
34	5.319	0.981	3.623	6.395	10.71	0.491	3.144	4.332	4.057
35	0.737	4.394	4.076	7.078	11.392	5.074	2.356	0.866	4.751
36	3.267	4.317	3.386	6.388	10.433	5.052	3.127	2.703	3.798
37	7.828	5.812	3.356	2.634	4.231	6.568	5.292	6.85	4.582
38	3.566	1.451	2.73	5.494	9.809	2.007	1.239	2.601	3.156
39	3.126	1.925	2.81	5.574	9.889	2.534	1.163	2.281	3.236
40	2.54	4.271	3.742	6.744	11.059	5.007	2.678	2.016	4.417
41	3.943	2.157	0.8	3.802	8.117	2.912	1.637	3.186	1.475
42	12.517	8.179	10.509	10.391	17.247	7.284	10.203	11.53	9.107
43	5.181	1.035	3.687	6.459	10.774	1.37	3.208	4.194	4.121

Table B.1 continued from previous page

44	2.798	2.876	1.945	4.947	9.262	3.632	2.135	2.041	2.62
45	2.139	2.801	2.993	5.765	10.079	3.273	0.556	1.277	3.668
46	2.442	2.555	3.106	5.878	10.192	3.235	0.669	1.645	3.781
47	4.75	4.644	5.629	8.393	12.708	5.2	3.539	4.492	6.055
48	4.31	0.63	3.205	5.977	10.292	1.32	1.957	3.319	3.639
49	2.007	4.818	4.4	7.401	11.716	5.517	3.338	1.848	5.075
50	2.201	4.703	4.213	7.215	11.529	5.402	3.148	1.904	4.888
51	7.023	5.981	7.521	10.293	14.608	6.537	5.812	6.788	7.955
52	0.749	4.295	4.294	7.296	11.61	5.174	2.391	0.767	4.969
53	2.479	2.808	3.009	5.631	9.946	3.403	0.927	1.322	3.442
54	4.61	0.464	3.116	5.888	10.203	0.798	2.637	3.623	3.55
55	2.371	3.846	4.853	7.617	11.932	4.402	2.427	2.258	5.279
56	2.462	3.29	3.672	6.444	10.758	3.899	1.235	1.96	4.347
57	2.709	2.31	3.081	5.852	10.167	2.866	1.125	1.85	3.514
58	4.648	2.632	0.71	3.244	7.559	3.388	2.113	3.671	1.402
59	2.385	3.371	4.378	7.142	11.457	3.927	2.887	2.733	4.804
60	6.817	7.965	7.105	7.268	11.582	8.771	6.749	6.253	7.657
61	2.424	3.086	3.271	6.05	10.364	3.558	0.841	1.287	3.779
62	1.84	3.415	3.323	6.291	10.606	3.887	1.169	0.691	3.953
63	2.304	3.228	2.829	5.831	10.146	3.837	1.813	1.578	3.504
64	2.233	3.285	3.011	6.013	10.327	3.841	1.364	1.076	3.686
65	10.722	6.4	8.73	8.612	15.468	5.505	8.424	9.735	7.328
66	3.231	2.435	1.955	4.726	9.041	3.24	1.821	2.228	2.388
67	8.186	3.864	5.766	5.648	12.504	2.968	5.778	7.199	4.364
68	8.139	4.466	5.436	5.318	12.174	3.691	5.448	7.006	4.034
69	1.239	4.952	5.14	7.912	12.226	5.42	2.703	2.205	5.767
70	3.208	3.81	3.244	6.246	10.226	4.509	2.763	2.543	3.591
71	3.203	3.681	2.801	5.803	9.783	4.488	2.888	2.639	3.148
72	2.983	4.031	3.151	6.153	10.133	4.838	2.843	2.419	3.498

Table B.1 continued from previous page

73	5.011	5.076	5.961	8.725	13.04	5.653	3.871	4.367	6.387
74	6.78	5.968	7.508	10.28	14.595	6.524	5.569	6.376	7.942
75	1.759	4.019	4.464	7.258	11.572	4.897	2.115	0.491	5.053
76	1.328	3.699	3.815	6.783	11.098	4.379	1.661	0.171	4.445
77	2.656	3.548	4.017	6.789	11.104	4.247	1.985	1.507	4.451
78	2.146	3.235	3.173	6.05	10.365	3.791	0.907	1.641	3.712
79	2.801	3.792	4.515	7.287	11.602	4.348	2.616	3.154	4.949
80	2.53	3.675	3.937	6.896	11.08	4.374	1.859	1.381	4.558
81	3.901	5.122	4.262	7.264	11.516	5.928	3.62	3.377	4.881
82	3.953	1.272	1.741	4.513	8.828	1.971	1.262	2.82	2.175
83	5.506	5.4	6.385	9.149	13.464	5.956	4.295	5.271	6.811
84	1.766	4.918	5.106	7.878	12.192	5.39	2.669	2.453	5.734
85	1.397	4.367	4.555	7.327	11.641	4.839	2.118	1.949	5.182
86	3.472	4.161	3.508	6.51	10.49	4.869	3.024	2.903	3.855
87	1.64	3.502	3.848	6.677	10.991	4.284	1.533	1.235	4.398
88	5.352	4.746	5.902	8.666	12.981	5.302	3.796	4.772	6.328
89	3.921	3.722	4.673	7.437	11.752	4.278	2.567	3.543	5.099
90	1.889	3.959	3.677	6.679	10.993	4.586	1.87	0.732	4.352
91	2.322	3.74	3.729	6.688	10.872	4.212	1.651	1.173	4.35
92	2.588	2.98	2.501	5.503	9.818	3.787	1.934	1.84	3.165
93	4.065	1.998	1.458	4.229	8.544	2.805	1.53	3.088	1.891
94	1.819	5.476	5.158	8.16	12.474	6.156	3.438	1.948	5.833
95	2.492	5.697	5.421	8.423	12.738	6.388	3.936	2.312	5.929
96	3.043	5.296	4.436	7.438	11.69	6.102	3.794	2.669	5.055
97	4.9	1.227	2.197	4.968	9.283	2.286	2.209	3.767	2.63
98	5.67	5.407	6.387	9.151	13.466	5.963	4.281	5.257	6.813
99	3.115	4.637	3.98	6.982	11.234	5.444	3.114	2.591	4.599
100	3.615	4.841	3.976	6.978	11.23	5.642	3.614	3.091	4.595
101	1.993	5.706	5.622	8.624	12.938	6.5	3.782	2.178	6.297

Table B.1 continued from previous page

102	1.157	4.87	4.786	7.788	12.102	5.664	2.946	1.342	5.461
103	1.435	5.148	5.064	8.066	12.38	5.942	3.224	1.62	5.739
104	2.287	6	5.916	8.918	13.232	6.794	4.076	2.472	6.591
105	3.375	2.873	1.516	4.518	8.833	3.628	2.353	2.618	2.191
106	2.536	2.926	2.607	5.575	9.89	3.625	1.473	1.379	3.237
107	3.041	3.108	2.228	5.23	9.394	3.915	2.377	2.284	2.759
108	3.388	3.459	2.579	5.581	9.561	4.266	2.728	2.635	2.926
109	3.133	3.2	2.32	5.322	9.302	4.007	2.469	2.376	2.667
110	3.7	3.611	4.562	7.326	11.641	4.167	2.456	3.432	4.988
111	6.79	3.117	4.087	3.97	10.826	4.176	4.099	5.657	2.685
112	1.45	5.328	5.516	8.288	12.602	5.796	3.079	2.416	6.143
113	3.868	4.943	4.083	5.405	9.718	5.749	3.728	3.304	4.702
114	7.936	9.084	8.224	8.387	12.701	9.89	7.868	7.372	8.776
115	4.974	1.456	2.004	4.776	9.09	2.515	2.437	3.995	2.438
116	1.48	5.901	5.825	8.827	13.141	6.604	3.883	2.446	6.5
117	0	4.723	4.743	7.745	12.059	5.408	2.691	1.268	5.418
118	4.723	0	2.912	5.785	9.992	1.059	2.426	3.528	3.339
119	4.743	2.912	0	3.272	7.587	3.712	2.437	3.986	1.964
120	7.745	5.785	3.272	0	6.865	6.484	5.209	6.767	4.007
121	12.059	9.992	7.587	6.865	0	10.799	9.523	11.081	8.813
122	5.408	1.059	3.712	6.484	10.799	0	3.093	4.421	4.146
123	2.691	2.426	2.437	5.209	9.523	3.093	0	1.624	3.112
124	1.268	3.528	3.986	6.767	11.081	4.421	1.624	0	4.616
125	5.418	3.339	1.964	4.007	8.813	4.146	3.112	4.616	0
126	4.556	2.482	4.022	6.794	11.109	3.067	2.656	4.125	4.456
127	1.637	3.378	3.724	6.553	10.867	4.271	1.409	1.111	4.274
128	0.766	3.957	4.57	7.342	11.656	4.85	2.133	1.485	5.197
129	3.229	4.381	3.521	6.523	10.775	5.187	2.879	2.665	4.14

Table B.1 continued from previous page

NODES	126	127	128	129					
depot	7.78	4.983	5.577	2.819					
1	3.578	1.369	1.954	2.038					
2	3.66	0.476	0.603	3.116					
3	4.286	1.366	0.496	3.028					
4	4.215	2.065	1.561	4.705					
5	4.459	1.998	2.493	1.311					
6	6.747	7.95	8.529	8.866					
7	8.522	6.061	6.834	4.245					
8	7.536	8.723	9.302	9.655					
9	3.492	1.251	0.674	3.844					
10	3.704	1.554	1.051	4.147					
11	3.561	1.61	1.107	4.25					
12	3.591	0.66	1.513	1.98					
13	4.389	4.148	4.937	4.118					
14	4.332	4.15	4.939	4.072					
15	1.676	2.277	2.856	3.503					
16	3.506	3.265	4.054	3.409					
17	4.346	2.196	1.692	4.836					
18	4.088	1.846	1.342	4.486					
19	4.917	2.612	1.728	4.246					
20	2.634	2.52	3.309	3.129					
21	3.174	0.962	0.825	3.602					
22	4.377	2.135	1.632	4.728					
23	1.331	2.736	2.668	4.32					
24	4.906	1.85	0.979	2.824					
25	5.226	5.443	6.232	6.008					
26	2.789	3.006	3.795	3.617					
27	3.585	1.109	1.962	1.531					

Table B.1 continued from previous page

28	3.934	1.473	2.246	1.167					
29	4.184	1.728	2.609	1.313					
30	4.078	1.622	2.503	1.419					
31	1.689	4.614	4.632	6.268					
32	4.437	1.976	1.739	1.747					
33	3.233	4.129	4.708	5.247					
34	3.162	4.182	4.761	5.098					
35	4.719	1.635	0.921	2.676					
36	5.139	2.678	3.451	1.056					
37	6.878	6.636	7.425	6.544					
38	1.556	2.429	3.008	3.655					
39	2.083	1.989	2.568	3.018					
40	4.704	2.229	2.724	0.763					
41	3.222	2.924	3.77	2.721					
42	10.177	11.38	11.959	12.296					
43	3.148	4.044	4.623	5.162					
44	3.942	1.779	2.66	1.576					
45	3.131	1.002	1.581	2.62					
46	2.762	1.305	1.884	2.989					
47	2.186	4.175	4.193	5.83					
48	1.931	3.173	3.752	4.313					
49	5.517	2.617	2.122	2.29					
50	5.084	2.699	2.385	1.772					
51	3.523	6.448	6.466	8.102					
52	4.636	1.734	0.864	2.783					
53	2.952	1.107	1.96	2.577					
54	2.577	3.473	4.052	4.591					
55	2.743	1.373	1.813	4.013					
56	3.448	1.325	1.904	3.303					

Table B.1 continued from previous page

57	2.415	2.024	2.151	3.194					
58	3.698	3.457	4.246	3.438					
59	2.268	1.848	1.828	4.488					
60	8.761	6.3	7.001	4.243					
61	3.274	1.287	1.866	2.335					
62	3.554	1.155	1.282	2.623					
63	3.386	1.363	2.216	1.715					
64	3.39	0.861	1.714	1.849					
65	8.398	9.585	10.164	10.517					
66	3.55	1.957	2.809	2.599					
67	5.861	7.049	7.628	7.687					
68	6.575	6.792	7.581	7.357					
69	3.799	1.649	1.146	4.221					
70	4.819	2.486	3.338	0.997					
71	4.798	2.595	3.387	0.992					
72	4.855	2.394	3.167	0.772					
73	2.732	4.507	4.454	5.735					
74	3.51	6.205	6.223	7.86					
75	4.564	1.458	1.976	3.125					
76	4.046	0.94	1.512	2.715					
77	4.25	1.52	2.098	2.98					
78	3.34	1.461	1.588	2.473					
79	1.847	2.269	2.244	4.108					
80	4.124	1.394	1.972	2.854					
81	5.842	3.381	4.085	0.741					
82	2.281	2.606	3.395	3.216					
83	2.942	4.931	4.949	6.586					
84	3.765	1.615	1.112	4.255					
85	3.527	1.064	1.017	3.704					

Table B.1 continued from previous page

86	5.159	2.75	3.602	1.261					
87	3.966	0.124	1.082	2.764					
88	2.288	4.432	4.795	6.016					
89	1.357	3.203	3.364	4.787					
90	4.135	1.405	1.983	2.163					
91	3.916	1.186	1.764	2.646					
92	4.096	1.568	2.459	1.601					
93	3.115	2.874	3.663	3.025					
94	5.731	2.717	2.003	3.644					
95	6.013	3.28	2.676	2.753					
96	5.925	3.464	3.227	0.915					
97	3.336	3.553	4.342	4.118					
98	2.949	4.917	5.113	6.501					
99	5.126	2.665	3.299	0.459					
100	5.626	3.165	3.799	0.754					
101	5.967	3.048	2.177	4.108					
102	5.131	2.212	1.341	3.272					
103	5.409	2.49	1.619	3.55					
104	6.261	3.342	2.471	4.402					
105	3.938	2.356	3.237	2.153					
106	3.648	1.356	1.998	1.924					
107	4.225	2.022	2.903	1.565					
108	4.576	2.373	3.254	1.214					
109	4.317	2.114	2.995	1.473					
110	2.033	3.092	3.143	4.676					
111	5.226	5.443	6.232	6.008					
112	4.267	2.025	1.521	4.281					
113	5.74	3.279	4.052	1.462					
114	9.88	7.419	8.12	5.362					

Table B.1 continued from previous page

115	3.564	3.781	4.57	3.925					
116	5.134	2.829	1.944	4.311					
117	4.556	1.637	0.766	3.229					
118	2.482	3.378	3.957	4.381					
119	4.022	3.724	4.57	3.521					
120	6.794	6.553	7.342	6.523					
121	11.109	10.867	11.656	10.775					
122	3.067	4.271	4.85	5.187					
123	2.656	1.409	2.133	2.879					
124	4.125	1.111	1.485	2.665					
125	4.456	4.274	5.197	4.14					
126	0	3.939	3.999	5.101					
127	3.939	0	1.069	2.64					
128	3.999	1.069	0	3.413					
129	5.101	2.64	3.413	0					

Appendix C

Table of Demands per Barangay

Table C.1: Demands per Barangay

Nodes	Demand	Node	Demand	Node	Demand	Node	Demand
Depot	0	**42**	7	**84**	2	**126**	4
1	5	**43**	2	**85**	4	**127**	11
2	7	**44**	9	**86**	7	**128**	11
3	4	**45**	13	**87**	14	**129**	5
4	10	**46**	13	**88**	5		
5	2	**47**	6	**89**	11		
6	5	**48**	5	**90**	13		
7	7	**49**	6	**91**	8		
8	2	**50**	11	**92**	10		
9	9	**51**	2	**93**	6		
10	8	**52**	1	**94**	12		
11	3	**53**	1	**95**	6		
12	5	**54**	14	**96**	8		
13	12	**55**	10	**97**	13		
14	11	**56**	10	**98**	4		
15	1	**57**	6	**99**	10		
16	8	**58**	12	**100**	12		
17	12	**59**	11	**101**	11		
18	9	**60**	3	**102**	5		
19	14	**61**	9	**103**	2		
20	10	**62**	10	**104**	3		

21	13	**63**	5	**105**	1	
22	12	**64**	7	**106**	4	
23	2	**65**	11	**107**	9	
24	14	**66**	4	**108**	2	
25	12	**67**	4	**109**	2	
26	3	**68**	8	**110**	12	
27	6	**69**	11	**111**	1	
28	3	**70**	8	**112**	2	
29	1	**71**	3	**113**	7	
30	1	**72**	6	**114**	4	
31	4	**73**	9	**115**	2	
32	13	**74**	3	**116**	5	
33	2	**75**	9	**117**	10	
34	5	**76**	3	**118**	7	
35	4	**77**	8	**119**	6	
36	1	**78**	4	**120**	11	
37	8	**79**	14	**121**	13	
38	9	**80**	1	**122**	6	
39	5	**81**	7	**123**	14	
40	12	**82**	12	**124**	3	
41	12	**83**	11	**125**	11	