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 have had many troubles tribulations in conducting this study however, I was not alone in bearing these problems. Firstly, I would like to acknowledge me adviser, Dr. Joel M. Addawe, for sharing his wisdom and dedicating his time to guide me during the conduction of this study. I would like to extend my appreciation to Prof. Lee Javellana who aided me in using the Matlab software. The model presented in this study was not solely my own but a product of the collaboration with my colleagues, Aaron Dumon, and Juancho Meneses. I would also like to acknowledge the General Services Office - Solid Waste Management Division of Baguio City for entertaining my requests and questions about the state of waste management in the city.

During the conduction of this study, I have fallen mentally and physically ill, therefore, I would love to extend my gratitude to my family and friends for their support in my pursuit to accomplish this study.

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.

In Baguio City, waste collection has been a main concern for the past years as the population continually increases. The waste collection problem was modeled as a Capacitated Vehicle Routing Problem which considers the total travel distance and fleet size. A hybrid PSO-GA algorithm was employed to obtain the set of routes that give the minimum amount of operational costs. The algorithm was tested under small scale problem instances and the number of collection sites involved were gradually increased until it failed to find any solutions. It was found that the hybrid PSO-GA algorithm was indeed able to solve small scale instances of the problem however, a large population size was needed for obtaining satisfactory results. The optimal set of vehicle routes needed to solve the full-scale problem was unable to be obtained due to the limitations of time and the inherent complexity of large scale vehicle routing problems.

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. There would be an abundant accumulation of waste in cities without waste collection and management services. In addition, our natural resources would dwindle without proper and efficient recycling methods. Human settlements would become unsustainable as the current lifestyle that is centered upon production and consumption leads to rapid generation of waste. Cities would become more susceptible to outbreaks and pandemics because accumulated wastes serve as breeding grounds for diseases and their carriers (i.e. rodents, mosquitoes, etc.). Given the effects of the lack of waste management services, all units of the community should establish their own waste management systems to cope with the problems that are brought about by large amounts of solid waste.

Waste, in this context, is defined as matter that is unwanted or unusable. Waste refers to matter which are discarded after its primary use or are deemed defective, or worthless upon production. There are various types of waste but in this study, we focus 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 (local government) to collect and manage these kinds of waste. 'Solid' refers to their physical state. A list of what can be considered as municipal solid waste is as follows:

- **Biodegradable waste** 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** are objects that can be processed for reuse. Examples of such objects are 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, with other substances 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** are objects that are composed of two or more constituent materials such as toys, tetra packs, clothing, fiber glass, etc.
- **Hazardous waste** are materials that can pose health risks such as paints, batteries, aerosol sprays, and fertilizers
- **Toxic waste** are poisonous substances such as pesticides, herbicides and fungicides
- **Biomedical waste** are objects that may have been used for medical procedures or sanitation. These objects may contain infectious materials such as expired pharmaceuticals, used medical equipment, used tissues, etc.

Waste management consists of all activities involved in handling waste from the source to its final disposal. This includes the collection, transportation, treatment, segregation, and disposal of waste. We focus specifically on waste collection which includes gathering, transporting, and disposing of solid waste.The waste collection process involves deploying a fleet of vehicles that collect and transport the waste from communities to facilities that receive, sort, and process the waste. Waste processing may be done in the form of incineration, rapid degradation, segregation, resource recovery, energy recover, etc.

Waste can be collected in several different ways. House-to-house collection, if possible, is done by individually visiting each house and collecting the garbage straight from the source. This can be done in small subdivisions and areas near disposal sites. Larger communities can settle for gathering waste at designated locations around the area or specifically at large containers called ”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 along the way. A schedule may be followed when the collection is done this way. Households can also volunteer to personally deliver their garbage directly to disposal sites. Local governments can get into a contract with private companies for waste collection services. These companies usually have specialized facilities and vehicles that can handle the job.[19]

Households can be asked to segregate their waste before collection. Waste can be sorted into different sets of 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 the efficiency and effectiveness of waste processing. Upon reaching disposal facilities, 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, cleaned, and re-purposed. Waste such as broken plastic frames or metals can undergo secondary use through remelting and remolding. Papers and cardboard can be recycled for a different use such as fire starters or insulation. 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 problems such as health-care, contamination, pollution, land use etc. On the other hand, incinerating waste produces greenhouse gases which may raise health care issues. We must also consider that there 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.

Waste management becomes better as we develop the methods and facilities needed. Improvements in waste management results in the development of other facets of the communities, however, this comes at a price. Urbanization and development comes with the production of large volumes of waste. Although economic developments and industrialization improves the standards of living, the increase in the total amount of disposable income encourages individuals to consume more goods and services, thereby resulting in an increase in waste generation. Major economies also run on an unyielding cycle of production and consumption which leads to a rapid increase in waste generation. According to the study conducted by the World Bank, called ”WHAT A WASTE: A Global Review of Solid Waste Management”, back in 2012, the 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, pollution, road blockades, rodent infestations, and disease outbreaks. 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 also impact an area's economy. Businesses that do not comply with the rules and regulations imposed by public health and safety get shut down. Uncollected waste can also result to a hazardous work environment that can endanger both 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.

Developed nations highly depend upon their waste management service providers to move waste out of urban districts and industrial sectors. These services help boost public health and provide a clean atmosphere for a conducive area of business. Given that waste collection and management is an important sector for any growing economy, there is a need to allocate funds for the development and maintenance of waste management services. 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. There is a high allocation of funds towards waste management because there are several requirements needed for a safe and efficient operation. First of all, the construction and maintenance of facilities needed for safely dealing with waste are expensive. Moreover, a large fleet of waste collection vehicles are needed to serve an ever growing population. Wages for both formal and informal employees are also covered in the budget. Formal employees are those who are known and legally registered under the company while informal employees are those who volunteer to help. The employees involved in waste management are not exclusively garbage collectors. There are multiple bodies at work in waste management. Statistical analysts and business managers are also needed for boosting efficiency not only in handling wastes but also in managing operational costs.

As discussed, waste collection services is an expensive service to develop and maintain. In order to quantify waste collection, we consider the costs involved. Costs in waste collection may involve staff salaries, vehicle purchases, construction of buildings and infrastructure, fuel expenditure, vehicle and equipment maintenance, land purchase, business contracts, environmental and health care expenditures, etc. Each of the collection options presented above also have different ways of computing cost. The population density, demographics, and coverage of the area also affects how much money is needed. It is therefore crucial to find ways to minimize the costs involved while not compromising efficiency. Therefore, we try to look into finding an operation that can be optimized. Specifically we try to generate the 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 availability. However, the use of such methods produce inflexible routes because they are instance-based data. In the recent years, with the development of commercial real-time 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. Moreover, there are various methods that can be used to solve these kinds of problems.

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 that allow each customer to be serviced by a vehicle in the fleet. The optimal solution is a set of routes that gives the minimum amount of traveling cost. Th VRP is a generalization of the **Traveling Salesman Problem (TSP)**. The TSP problem description is as follows: a salesman is tasked to visit a number of cities during a single trip. He/she must begin and end at the same city. Each city is linked to all other cities. There may be a number of ways to travel from one city to another such as by plane, vehicle, boat, etc. Each of these options are associated with a cost that represents how "expensive" it is to travel from one city to another using that means of transportation. The salesman wants to minimize the costs involved in traveling the entire trip. This can be done by taking on the set of transportation options that give either the least amount of distance, time, expenses or a combination of the three. 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 vehicle routing problem is typically the same problem however, there are more than one salesman (vehicle). The load of the single vehicle is distributed among the fleet in order to save other costs (i.e. time) and maximize man power. VRP is a combinatorial problem that deals with arranging multiple customer points along one or more path(s) that gives the smallest amount of transportation cost while satisfying known constraints. When we ignore expenses, VRP is typically 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 are assumed to 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 depositing their load at disposal sites. Vehicles start at a depot where they are parked when inactive. They are then dispatched 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 its 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.

In order to solve municipal waste collection VRP, we use the hybrid Particle Swarm Optimization-Genetic Algorithm (PSO-GA) proposed by Harish Garg[14]. The hybrid PSO-GA algorithm is a heuristic approach used in solving constrained optimization problems. Garg used the algorithm in solving engineering designs problems such as the design of a pressure vessel problem and the welded beam problem. These problems are constrained due to the nature of the materials used and physical laws involved. VRP can also be considered a constrained optimization problem wherein the constraints are imposed by vehicle capacities, company policies, time limits etc. Hence we approach the problem with the algorithm. The next sections of this paper are as follows. Chapter 2 is a section dedicated to the different terms used throughout this document. Chapter 3 involves the discussion of related studies conducted by researchers in the previous years. Chapter 4 is a discussion of the method used in order obtain the set of routes. Chapter 5 is where the results from the tests done with the method in chapters 4 is analyzed. Chapter 6 is where the results are summarized and conclusions are stated.

1.1 Background of the Study

According to the National Solid Waste Management Status Report (2008-2014) by the National Solid Waste Management Commission (NSWMC) under the Department of Environment and Natural Resources[26], about $37,000$ tons of municipal solid waste (MSW) are produced in the Philippines. Based on 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, 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[27]. 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 about 400 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 contributions of the residential, institutional, and industrial sectors are 35.16%, 3.53%, and 0.86% respectively. Baguio City is essentially 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 management 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 by the NSWMC 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.[33] 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 on January of the same year. 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.[34] The 14 waste collection vehicles are responsible for servicing the 129 barangays (villages) inclusive of the Central Business District. Vehicle compartments are somewhat 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 and 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 reached. Waste is returned to the ERS-MRF for final sorting. The residual waste is then transported to the Garbage Transfer Station at Barangay Dontogan where it is placed before it is finally transported to Capas, Tarlac where for final disposal. 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 its part to reduce the carbon footprint 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 styrofoams 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.[30]

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

1.2 Statement of the Problem

We want to find a way to reduce the operational costs of waste collection in Baguio City. It was observed in What a Waste[19] that the cost of waste collection is large for any developing country. In parallel, Baguio City, for over 10 years, has spent over 1 billion Philippine Pesos (Php) for hauling the city's solid waste to the sanitary landfill in Capas, Tarlac.[21] This involves covering the costs in 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 was spent for hauling and tipping fees in Capas, Tarlac; this amount also covers the expenses on personnel, garbage trucks, and other operating expenses. The city council, like any other, aims to reduce annual solid waste management expenses so that the budget 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 involved in Baguio City's waste management can be optimized in order to reduce the costs of operation. We also want to explore the capabilities of the hybrid PSO-GA algorithm proposed by Harish Garg[14].

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 into 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 travel cost while also determining the minimum number of waste collection vehicles required for completing the task.

The secondary objective of this study is to test the effectiveness of the hybrid PSO-GA algorithm on vehicle routing problems which is considered as a constraint optimization problem.

1.4 Significance of the Study

The researcher is hopeful that the results from this study will allow the determination of the vehicle routes that may minimize the operational costs 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 may 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.

1.5 Scope and Limitation

The study is limited to generating a set of routes involved in waste collection at Baguio City. In order to model the waste collection problem, each of the barangay halls 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 locations of the specific collection sites vary depending on a number of variables such as accessibility, road conditions, weather, and public health regulation. The ERS-MRF at Barangay Irisan was identified as the location where each waste collection vehicle

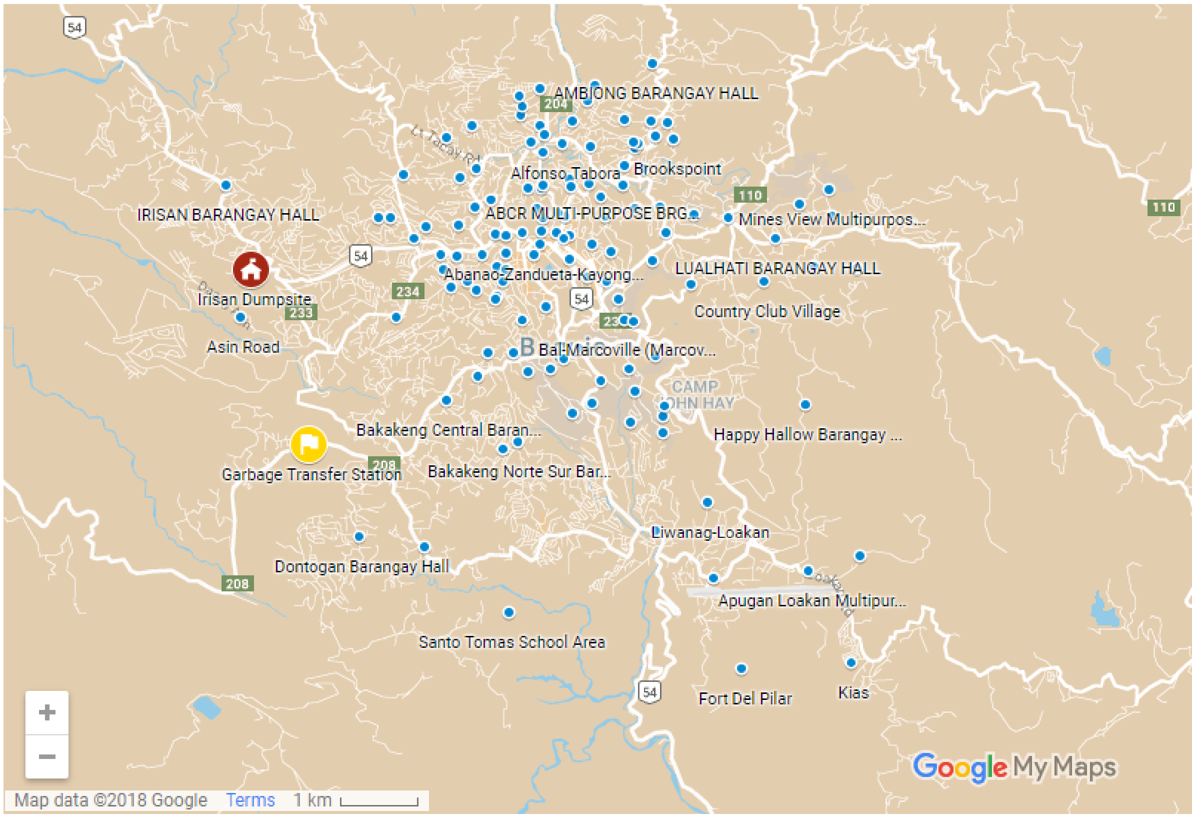


Figure 1.1: A Map Showing the Locations of the 129 Barangays and the Irisan ERS-MRF

begins and ends its daily operation. Given that there exists no hourly 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 each pair of barangay halls as well as the distances between each barangay hall and the ERS-MRF 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

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 used 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 equations. 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 satisfied.

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

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 to solve the problem. Optimization problems aim to obtain the minimum and/or maximum value of certain properties related to improving 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 change or are somewhat adaptive 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 its 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. **Chromosomes** contain the genes which dictate the attributes of the individual. They tell how the body is to be built and how it functions. 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.

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 an algorithm to search solutions in parts of the subspace it has not yet taken into consideration.

17. **Exploitation** is the capability an algorithm to utilize known data in searching for solutions in the search space.

18. 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 $A = \{1, 2, 3\}$.

19. 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.

20. **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.

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

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

23. 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.

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

25. 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.

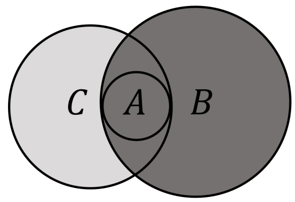


Figure 2.1: A Venn Diagram Showing the Relationship of A, B and C

26. 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.

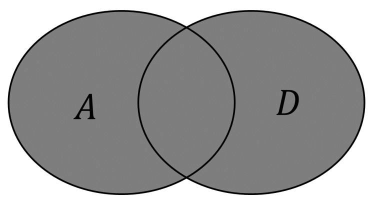


Figure 2.2: A Venn Diagram Showing $A \cup D$

27. 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.

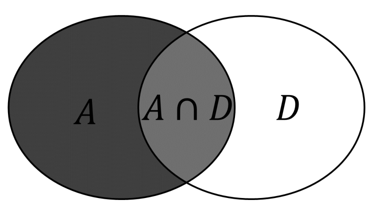


Figure 2.3: A Venn Diagram Showing $A \cap D$

28. 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.

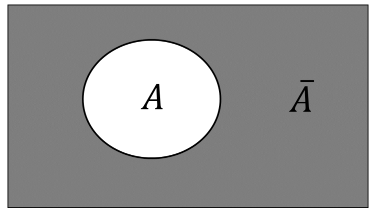


Figure 2.4: A Venn Diagram Showing $\bar{A}$

29. 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.

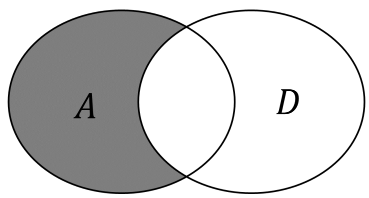


Figure 2.5: A Venn Diagram Showing $A - D$

30. 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 .

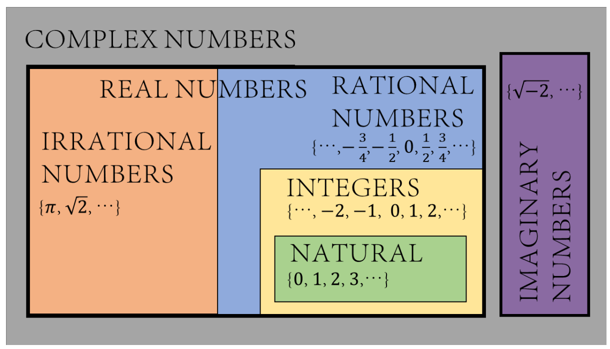


Figure 2.6: A Venn Diagram Showing the Relationship of Number Sets

31. 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.

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

33. 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 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.

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

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

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

37. 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, $(3, 5, 7, 9, \dots)$.

38. 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".

39. 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:

    - $(1, 2, 3)$
    - $(1, 3, 2)$
    - $(3, 1, 2)$
    - $(3, 2, 1)$
    - $(2, 3, 1)$
    - $(2, 1, 3)$

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

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

42. 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 $\overleftrightarrow{AB}$.

43. 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.

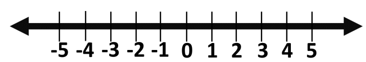


Figure 2.7: A Real Number Line

44. 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.

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

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

47. 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.

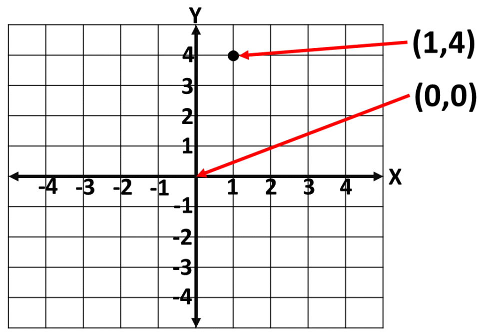


Figure 2.8: A Coordinate Plane

48. 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 $< x_1, x_2, x_3, \ldots x_n >$ where each component x_i is the absolute numerical distance between two points in dimension $i \in (1, 2, \ldots, n)$. 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 $i \in (1, 2, \ldots, n)$.

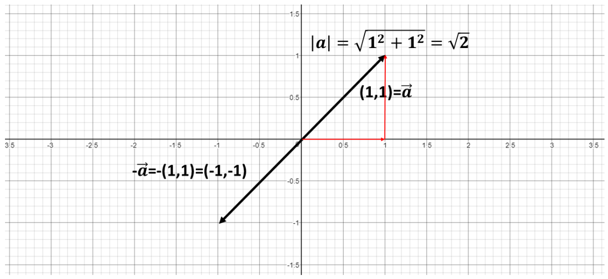


Figure 2.9: Vectors in a Coordinate Plane

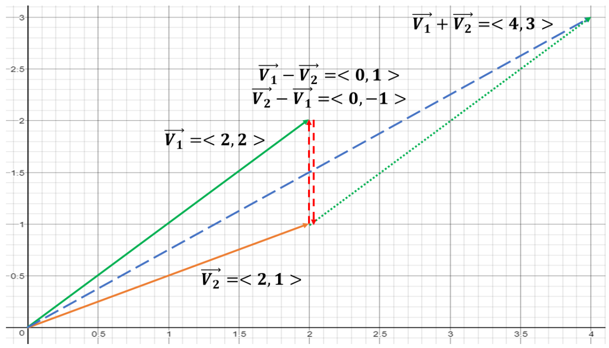


Figure 2.10: Adding and Subtracting Vectors (Coordinate Plane)

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

50. 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.

51. **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.

52. 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.

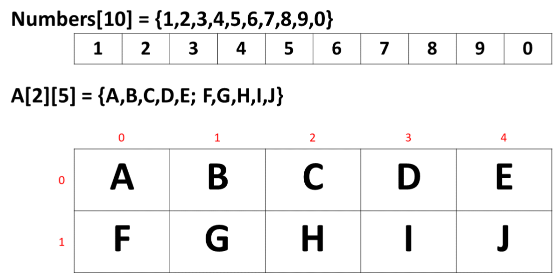


Figure 2.11: Visual examples of arrays

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

    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

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

An example is

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

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

54. 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

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

55. A matrix is said to be **symmetric** if and only if the matrix M is equal to its transpose M^T. Given by $M = M^T$. An example is

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

56. 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 $G = (V, E)$.

57. 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).

58. **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.

59. 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)).

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

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

62. 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').

63. 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.

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

65. 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 $\exists(u, v) \in E\ \forall u, v \in V$).

66. A graph is said to be a **weighted graph** if numbers are assigned to its edges. These numbers are called **weights** or **costs**.

67. 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.

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

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

70. 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.

71. 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.

72. 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.

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

74. 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

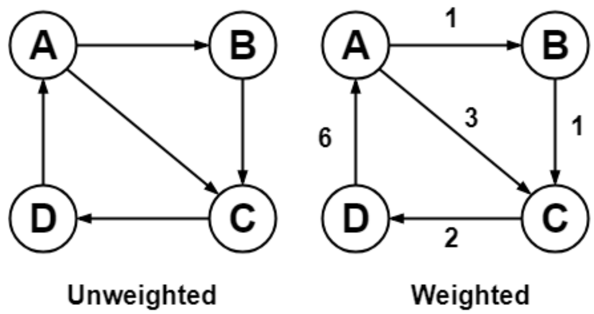


Figure 2.12: Sample Graph

has the adjacency matrix:

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

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:

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

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

75. 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.

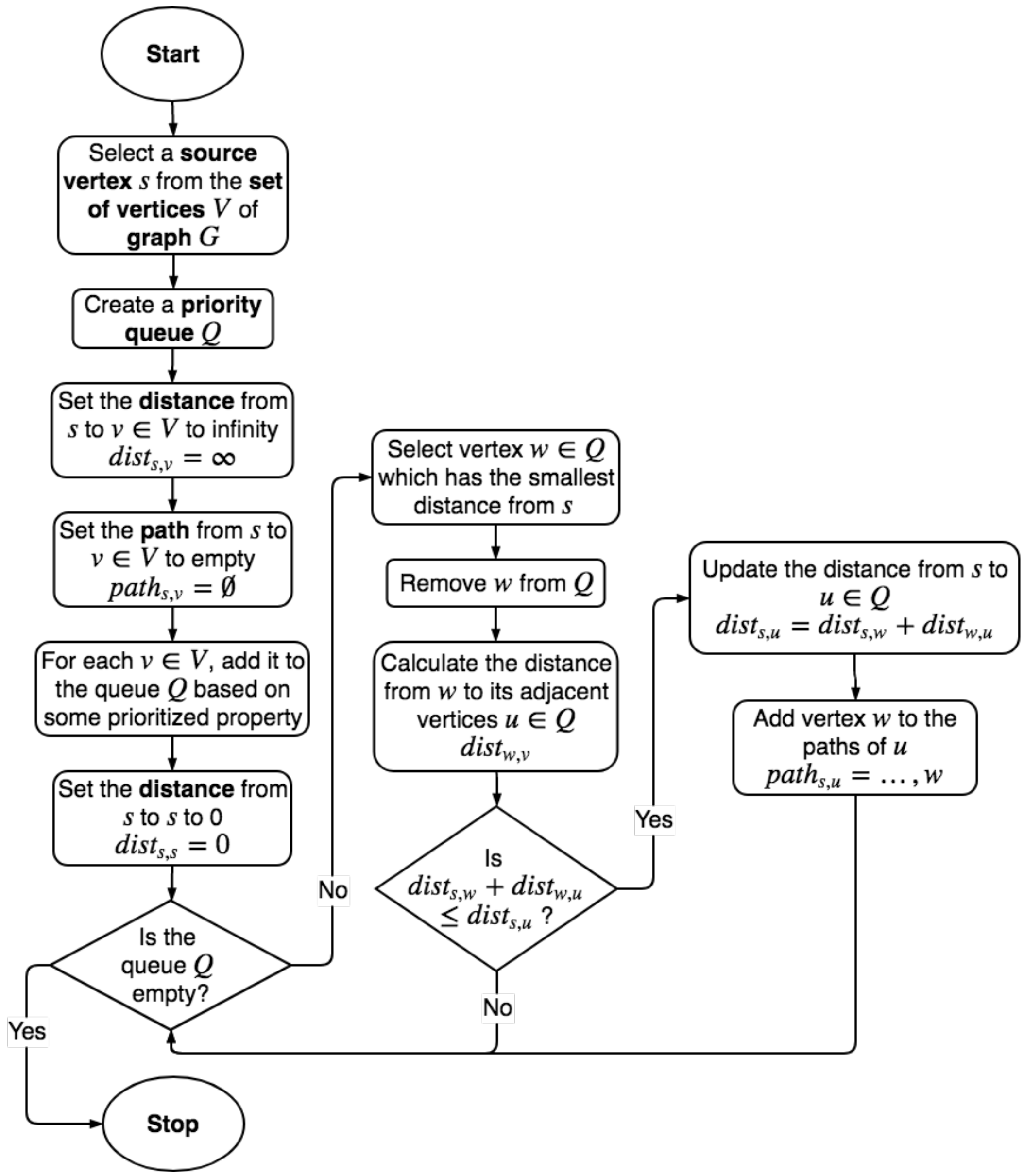


Figure 2.13: Flowchart of the Dijkstra Algorithm

(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.

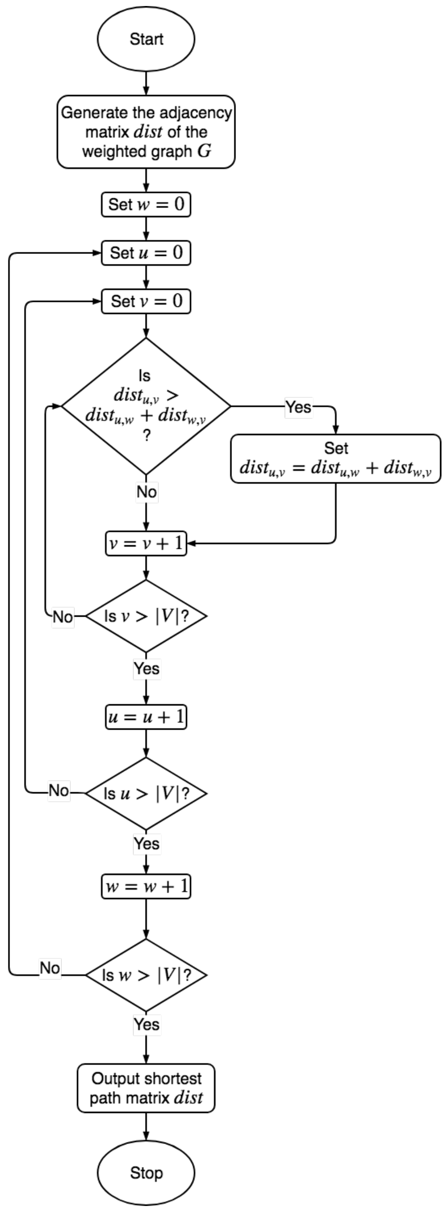


Figure 2.14: Flowchart of Floyd-Warshall Algorithm

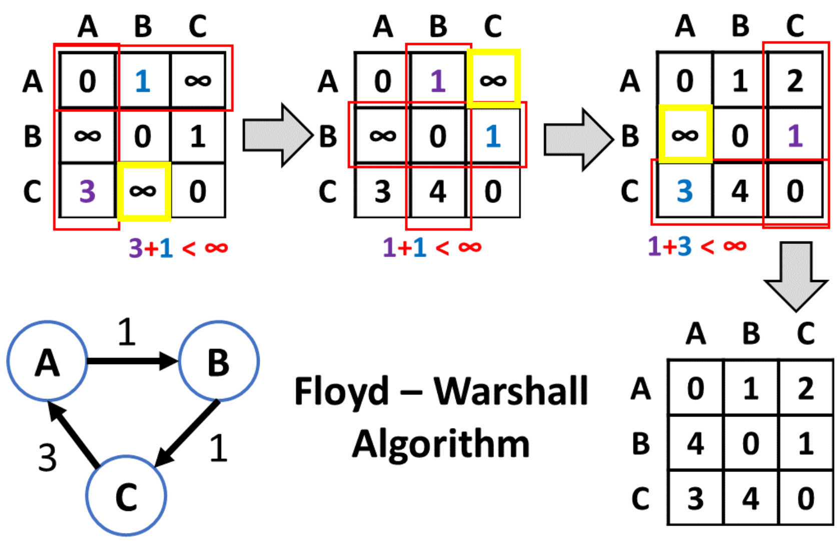


Figure 2.15: Floyd-Warshall Algorithm Example

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.

76. 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.

77. 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 $V = 0, 1, 2, \dots, n$ 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 its 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.

78. **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.

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

80. 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.

81. 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.

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

83. **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.

84. **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 $\mathbb{Z}$.

85. 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.

86. 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,

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

and q inequality constraints,

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

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

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

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

87. **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.

88. **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 chapters, 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 these kinds of problems. They give a brief description of the process of how each method solves the problems presented. This paper was used as an overview of past studies and serves as a basis on what methods has already been utilized in solving VRP.

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.Dantzig and Ramser stated that the traveling salesman problem is merely concerned with determining the shortest possible route which passes through each of the n given cities exactly once. Assuming that there exists some link that directly connects each pair of cities to and fro, 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 of fuel to be delivered to that service station. The vehicle is imposed to have the ability to carry only a total amount of C units of 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". These problems sets are for benchmarking solution methods that are 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. No vehicle is allowed to service a customer outside the allocated time window.

The first route construction algorithm is called the 'savings' heuristics which starts out with each customer having dedicated routes. This means that each vehicle exclusively services a customer. 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 of routes saves the most amount of cost among all other pairs. 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 is nearest, in terms of distance or time, to the current location of the vehicle. The process is repeated until the vehicle's schedule is full. 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. A vehicle's initial 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 a node is inserted, the succeeding nodes 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 node clustering.

In 2000, Son[35] utilized a Chaotic Particle Swarm Optimization (CPSO) algorithm to generate routes and schedules of the different waste collection vehicles at Danang City Vietnam. CPSO is discussed later in this chapter. The CPSO obtained data on the roads and waste collection facilities through 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 observations within 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 tricycles and forklifts were used to directly collect waste from household and deliver them to an intermediate disposal site. The hook-lift is then used to transport the gathered waste to its final destination. 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.[28] 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 considerably improved routes from those previously 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 (TSP) 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. Both PSO and GA are discussed late in this chapter. 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. The whole process of the PSO algorithm is followed completely at each iteration while the mutation mechanism only occurs when the population's total health is low. A 'healthy' particle is one that changes its 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 it can be said 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.

Similarly, Lu et.al [22] also developed a hybrid PSO algorithm wherein 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 for 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

$$\begin{array}{cc} \text{Nodes} & \begin{array}{cccc} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 0} \end{array} \\ \text{Particle} & \left[\begin{array}{cccc} 3 & 4 & 1 & 2 \end{array} \right] \end{array}$$

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 in the population, a section of the position vector is selected and removed in each particle. The extracted 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[38] conducted a case study wherein they investigated the refuse collection of a public company (URENCO) in five urban districts of Hanoi, Veitnam. The aimed to improve the company's daily operation, particularly 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

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

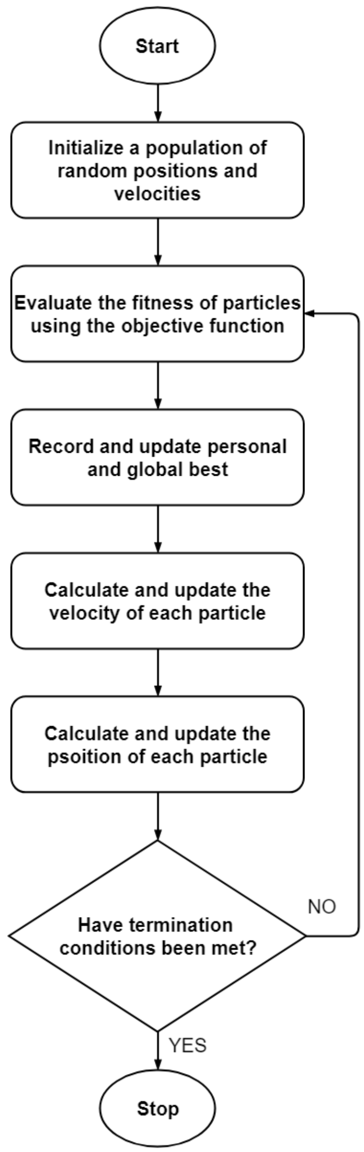


Figure 3.1: Flowchart of the PSO Algorithm

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 $x_{i,j}$ can represent while b_j is the highest number that each $x_{i,j}$ can represent.

Each particle's personal best value and location are recorded as *pbest* and $pbest_{i,d}$. In every iteration, the recorded *pbest* value is compared to the corresponding particle's current fitness value. *pbest* and $pbest_{i,d}$ are updated if the fitness value of the particle in the current iteration is better than that of the recorded one. *pbest* and $pbest_{i,d}$ act as a memory of where the particle was last at its best. Another pair of values and locations are recoded which are the population's 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 $pbest_{g,d}$. This pair serves as a memory of where the currently known optimal 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 changed in step 14. As we can see, there are many variables involved in the equations. They will be discussed in the next subsections.

3.2.1 PSO 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 instructions. 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[32], 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 dimensions. 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 dimension 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 every time 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 ($v_{i,d}$) 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 $pbest_{i,d}$ and $pbest_{g,d}$. In addition, without it, the search space will shrink and never grow since it will only move toward the centroid of its recorded locations $pbest_{i,d}$ and $pbest_{g,d}$. The two terms $c_1 \cdot rand() \cdot (pbest_{i,d} - x_{i,d})$ and $c_2 \cdot rand() \cdot (pbest_{g,d} - x_{i,d})$ 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

$$v_{i,d} = v_{i,d} \cdot w + c_1 \cdot rand() \cdot (pbest_{i,d} - x_{i,d}) + c_2 \cdot rand() \cdot (pbest_{g,d} - x_{i,d}) \tag{3.2}$$

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.[31] 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[39] as

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

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 [39] 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.

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 genetics. 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 implements 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

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
        (Acceptance and Replacement) 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

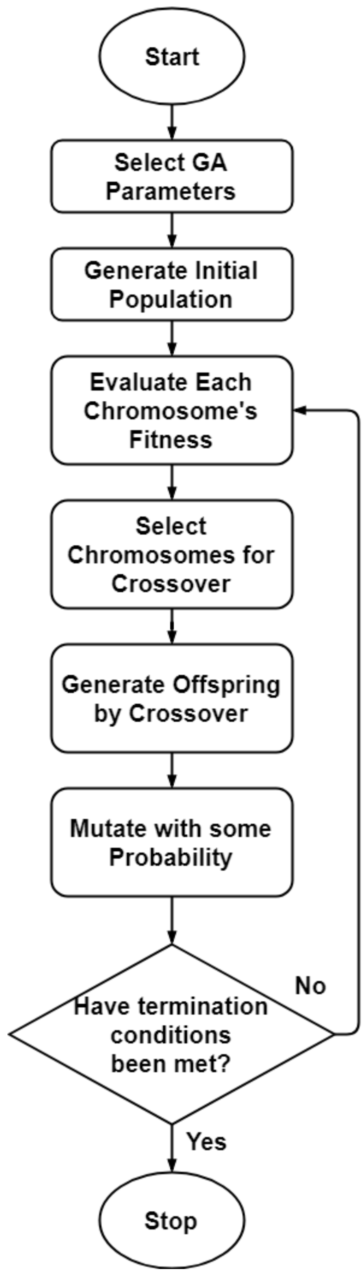


Figure 3.2: Flowchart of the GA Algorithm

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.[29] 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 mathematical equation that the algorithm is trying to optimize. The word "fitness" is taken from the evolutionary theory. The function tests and quantifies how 'fit' each potential solution is with respect to the problem.[3] It is important to note that the fitness function is a large factor in problem solving using GA. The fitness function must be able to adopt 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 the persistence and propagation of fitter genes in the current population onto the next generation. Fitter genes are determined by the fitness value of the individual. This is seen in nature wherein creatures that are well-adapted to the current environment would live longer than those that are not. In the animal kingdom, females are more likely to reproduce with males who can provide food, shelter, safety, and healthy offspring. Hence, the less fit individuals in the population have lower chances of reproducing and spreading their inferior genes. In GA, we want to retain individuals that are more capable of obtaining the optimal solution.

The selection process can either be repeating or non-repeating. The selection process is 'repeating' if it allows re-selection of previously selected members. Selection is 'non-repeating' if it does not allow re-selection of members. Non-repeating allows retention of other possibly 'good' genes (these are 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 almost the same 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 wheel selection and elimination selection. Fitness proportionate selection also called the roulette wheel selection is done by assigning probabilities to individuals in the population based on their fitness. Fitter chromosomes should have a higher chance of being selected while inferior chromosomes are less likely to be selected. The sum of the probabilities of assigned to each chromosome in the population should be equal to 1. This makes it similar to a roulette wheel. The wheel is 'spun' through the random number generator and the corresponding member mapped to the portion of the wheel where the number 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.

Recombination or Crossover

Crossover 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. This mechanic is similar to the biological process called 'heredity'. The crossover 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

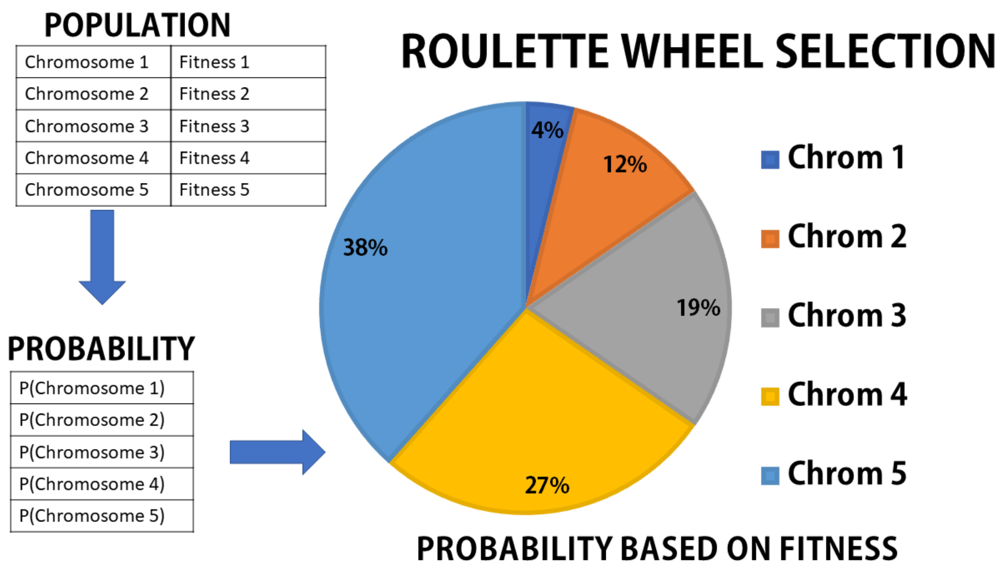


Figure 3.3: Roulette Wheel Selection

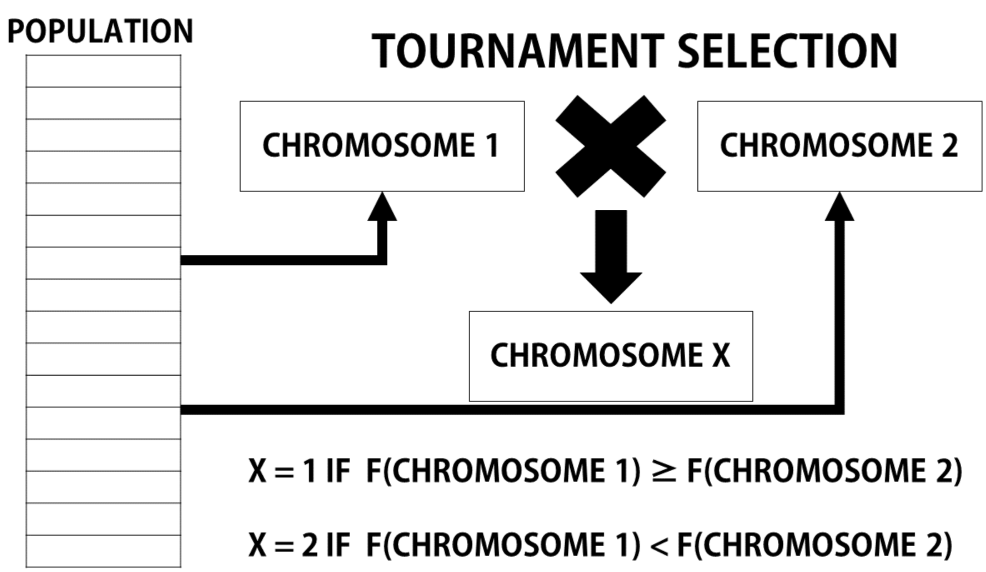


Figure 3.4: Simple Elimination Selection

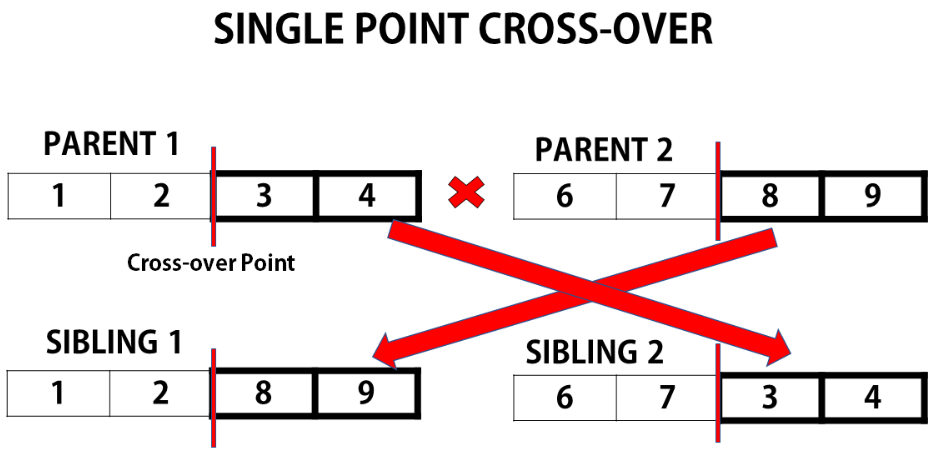


Figure 3.5: Singe Point Cross-Over

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 crossover 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. Crossover only happens under a probability. If crossover does not occur, there are not genes swapped and the offspring produced are the exactly the same as their parents. The crossover chance and number of crossover points are determined by the implementor after some tests.

Mutation

The mutation mechanism is the process wherein a set of genes from some members in the population are either replaced by a completely new value or are interchanged. This allows for exploration on possibly 'unexplored' areas in the search space. It reduce the chances of the population converging to a local optima. Local optima are the best solutions that can be found for a certain area in the search space but are not the most optimal of solutions in the entire search space. Like the crossover mechanism, mutation also occurs for only some probability. In nature, mutation is a rare occasion hence the mutation chance must be low (usually a probability of 0.02). A visual representation is

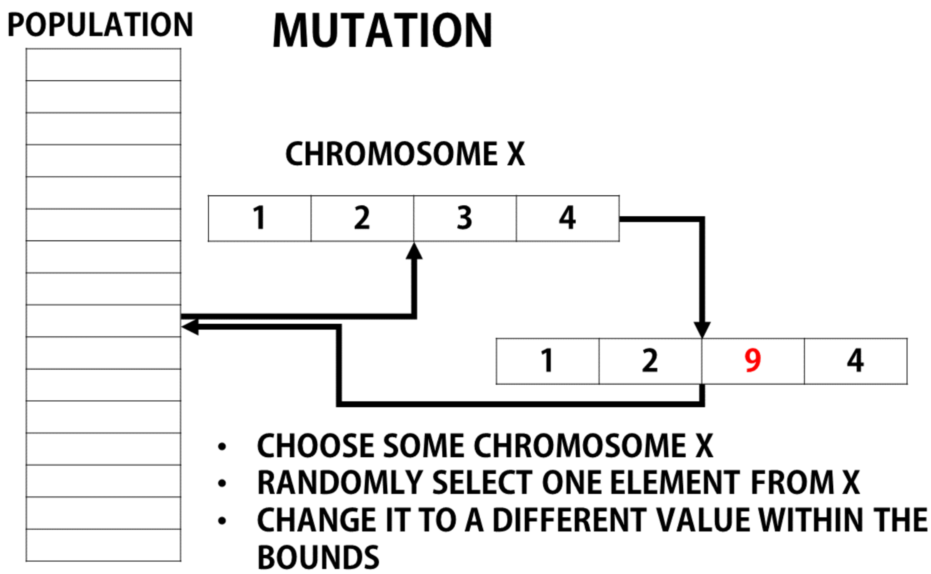


Figure 3.6: Mutation in GA

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 to be mutated is not limited to only one. It is possible to can change a few more genes but the results should only be a minor change. Mutation is defined 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 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 and Replacement

Under acceptance, we evaluating whether or not the newly generated chromosomes 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 we replace the parents, otherwise we keep the parents. This step is usually done after crossover to check if the siblings generated will replace members in the population based on their fitness. The retention of old individuals in the population is called being 'elitist' since it keeps only the fitter members of society to move forward.

Termination Conditions

Testing is done through keeping track of the best fitness value at each generation. If the fitness value 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. Real-worlds problems are always subjected to constraints. For example, if we were to create 3d models of a cube with a single piece of 10x15 inches cardboard. Given that we 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? We try and think of the best ways of cutting the cardboard so that we can get the best results. That was just a simply problem. 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 swarms. 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, its fitness value is unchanged but if it does not satisfy the conditions, its 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 currently known in the population. It only occurs for GA through Crossover 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 Crossover 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 crossover 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,

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

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

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

$$PSOx_i = PSOx_i + v_i \quad (3.8)$$

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, crossover 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 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,

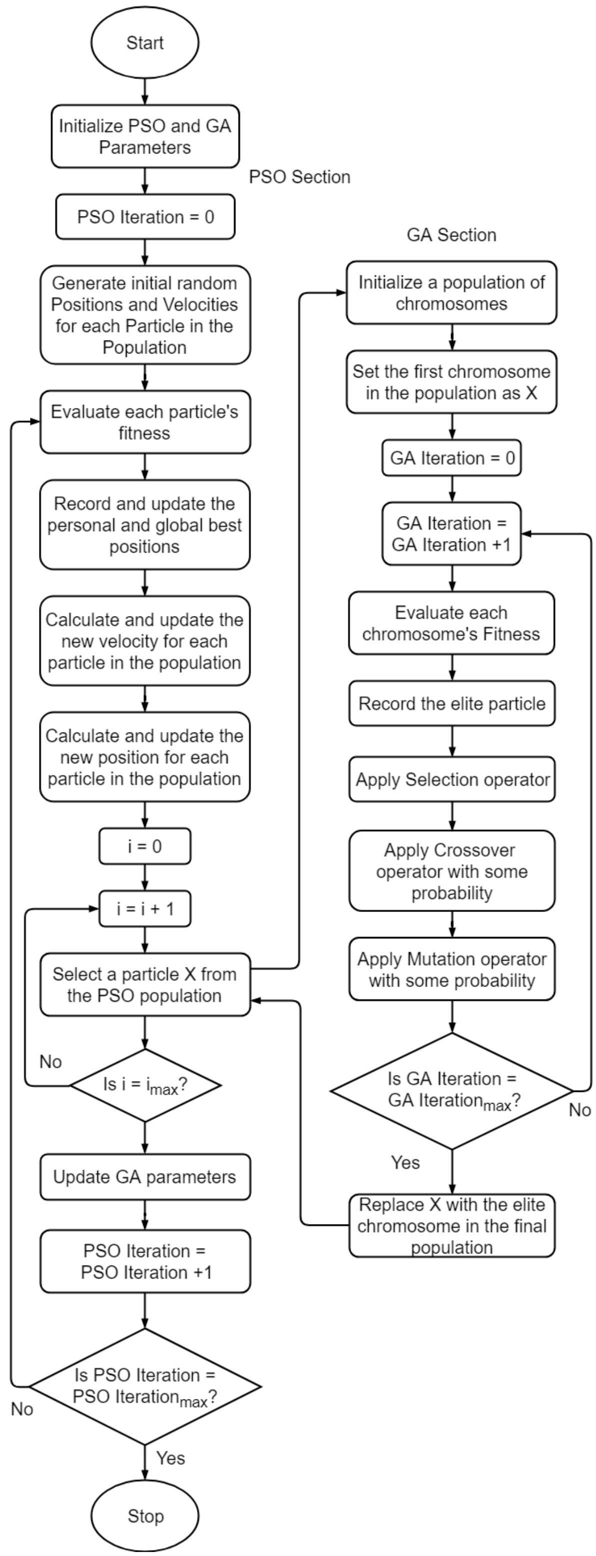


Figure 3.7: Flowchart of PSO GA Algorithm

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.

4.1 Problem Model

We now develop the specific model for the Baguio City waste collection vehicle 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 one of three shifts; morning, afternoon, and night.
- Each driver is assigned a 5-day work schedule among 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 specific time windows are alloted to each collection site due to variability of traffic, road availability, weather conditions, and quantity of waste.

4.1.1 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 body;
7. The demand at each collection site should be less than or equal to 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.

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

where

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

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.

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

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

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

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 $K = \{k_i\}, i \in 1, 2, 3, \dots, m$ 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 its 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 twice 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 its 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 not only the total amount of travel cost but also the fleet size (number of vehicles used). We already know that minimizing travel cost is about selecting the best set of road segments 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:

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

Where α_1 is the constant which converts distance to cost, α_2 is the constant which converts the total volume of 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:

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

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

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

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

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

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

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

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

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

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

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

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. We will know how many times v_i is visited by all vehicles since the values of each $X_{i,j,l}$ is 1 if vehicle k_l moves from vertex v_i to v_j and 0 otherwise. 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 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 the 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 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

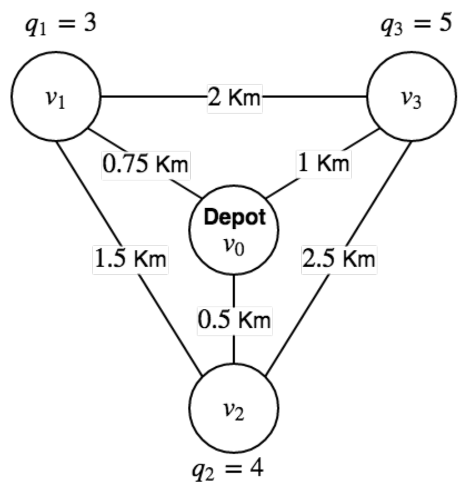


Figure 4.1: Graph of the Basic Example

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 regular VRP. The problem would be as reduced to finding the shortest connections between nodes by using 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 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, and then returning to the depot after collection. The edges they used are given as follows.

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

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

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

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

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

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.

4.2 Algorithm Implementation

In this section, we discuss how the PSO-GA was implemented. We first identify the method of encoding each particle or chromosome in the population. Each particle or chromosome is an array 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 numbers 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:

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

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

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

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 depot nodes at each end 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 hybrid PSO-GA algorithm implementation. An initial population of size PSO_{PopNum} is created by generating a set of random feasible 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 $N = 2n - 1$ 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 represented 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(4.4) 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 even though the vehicle 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 again 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 represent. 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 1, $GA_{CurrItr} = 1$.

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

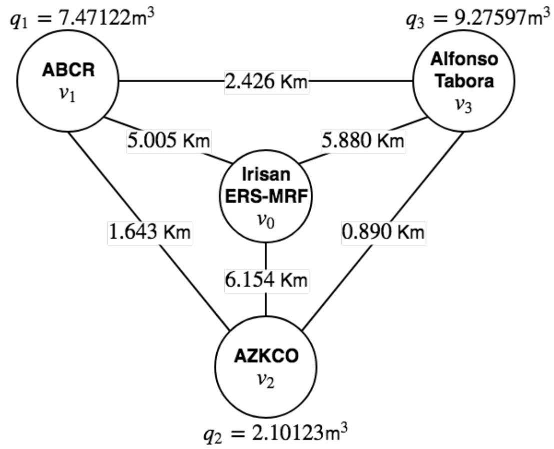


Figure 4.2: Graph of the 4 Nodes Small Scale Example

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 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.

4.3 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 constants $\alpha_1 = 12.474$, $\alpha_2 = 400$ 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 cubic meters because most of the vehicles are assigned about 7 to 8 areas to service each day. $\lceil 12/8 \rceil = 2$. The maximum is set to 12 cubic meters because this is stated in the assumptions of the model. Also note that even if there 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

$$\begin{aligned} v_0 \to v_1 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 = &\quad 12.474 \cdot 34.078 + 1,891.719333 = &\quad \text{Php } 2,316.808305 \\ v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_0 = &\quad 12.474 \cdot 24.562 + 1,051.719333 = &\quad \text{Php } 1,358.105721 \\ v_0 \to v_2 \to v_3 \to v_0 \to v_1 \to v_0 = &\quad 12.474 \cdot 22.934 + 1,051.719333 = &\quad \text{Php } 1,337.798049 \end{aligned}$$

The rest of the possible solutions violate the capacity and servicing constraints.

We test our PSO-GA algorithm using this small scale graph of the problem. The following parameters were used. These parameter values were selected based on the study of Garg[14].

- PSO Population Size = 5, 10
- PSO Maximum Iterations = PSO Population Size $\times 10$, " $\times 15$, " $\times 20$, " $\times 25$, " $\times 30$, " $\times 35$, " $\times 40$, " $\times 45$, " $\times 50$
- 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 $\times 0.1\rceil$
- GA Initial Population Size = 10
- GA Final Population Size = 5

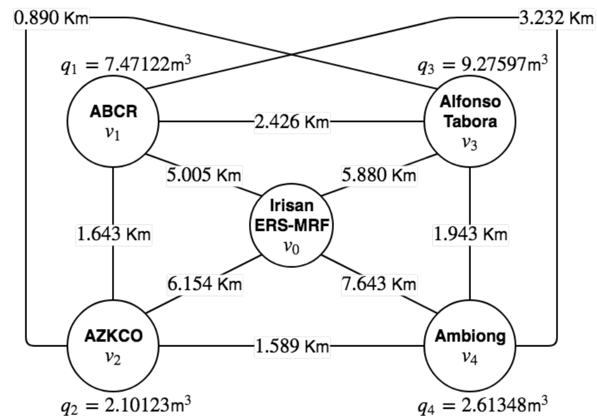


Figure 4.3: Graph of the 5 Nodes Small Scale Example

- GA Minimum Iterations = 10
- GA Maximum Iterations = 15
- Acceptance Threshold = 1×10^{-5}

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

$$v_0 \to v_1 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 \to v_4 \to v_0 = \quad \text{Php } 3,260.369869$$
$$v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_0 \to v_4 \to v_0 = \quad \text{Php } 2,301.667285$$
$$v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_0 \to v_3 \to v_0 = \quad \text{Php } 2,302.914685$$
$$v_0 \to v_3 \to v_4 \to v_0 \to v_1 \to v_0 \to v_2 \to v_0 = \quad \text{Php } 2,275.920949$$
$$v_0 \to v_2 \to v_3 \to v_0 \to v_1 \to v_0 \to v_4 \to v_0 = \quad \text{Php } 2,281.359613$$
$$v_0 \to v_2 \to v_4 \to v_0 \to v_1 \to v_0 \to v_3 \to v_0 = \quad \text{Php } 2,268.087277$$
$$v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_4 \to v_0 = \quad \text{Php } 1,317.218365$$
$$v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_3 \to v_0 = \quad \text{Php } 1,323.904429$$

We employ the PSO-GA algorithm to the larger scales of the problem. The 129 barangays are given in table A.1. All the distances used are seen on table B.1. The demand at each barangay is given at table C.1. We test the algorithm for 8, 11 and 16 nodes. The chosen barangays are still in the alphabetic order as in table A.1. We keep the same parameters except for PSO population size and maximum iterations. We set both parameters to 100 and 1000 respectively. However during the tests for for 11 and 16 nodes, the algorithm failed to find feasible solutions, hence we extend the PSO population size and maximum iterations to 250 and 25000 respectively.

Moreover, we test the PSO-GA algorithm for the full problem size of 130 nodes. The parameters were set the same except for PSO population size and maximum iterations. These were set to 100, 250 and 1000, 2500 respectively.

Chapter 5

Results and Discussion

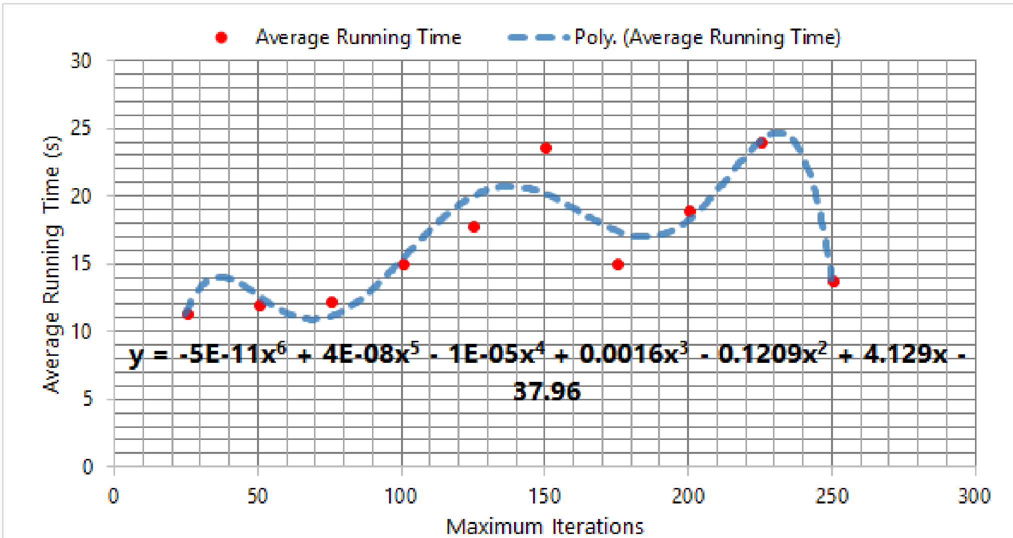
We now present the results of the small scale tests. The summary of the results in the small scale test with 4 nodes (3 barangays) is seen on table 5.1 while the test with 5 nodes (4 barangays) is seen on table 5.2.

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

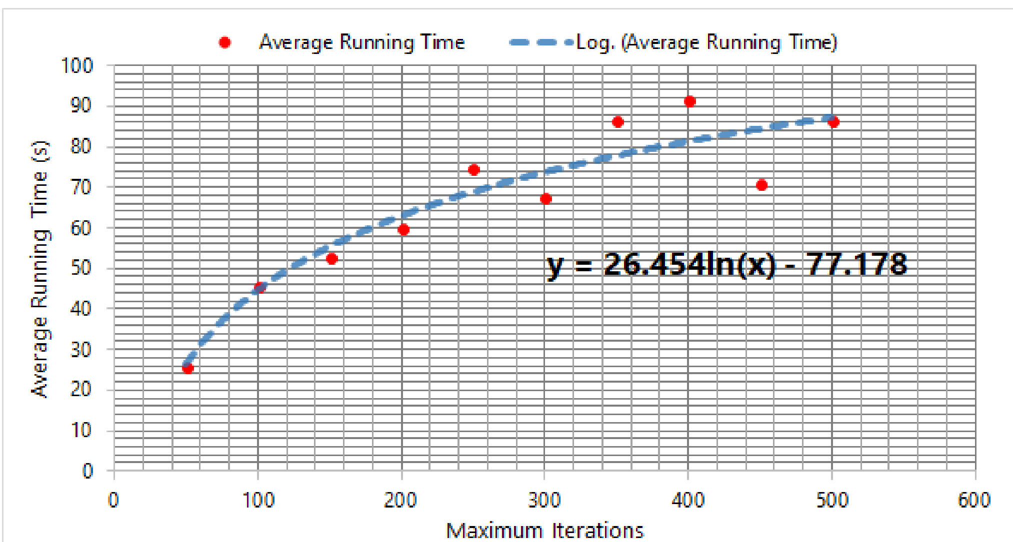
Pop. Size	**Max Itr.**	**Best Value**	**Succ. Rate (%)**	**Mean**	**Ave. Run Time (s)**	**Std. Dev.**
5	25	1337.798049	50.00	1339.828816	11.475944	6.421850
	50	1337.798049	100.00	1337.798049	12.044444	0.000000
	75	1337.798049	90.00	1339.828816	12.263487	6.421850
	100	1337.798049	100.00	1337.798049	15.040479	0.000000
	125	1337.798049	90.00	1339.828816	17.886521	6.421850
	150	1337.798049	100.00	1337.798049	23.639049	0.000000
	175	1337.798049	100.00	1337.798049	15.077499	0.000000
	200	1337.798049	90.00	1339.828816	18.962657	6.421850
	225	1337.798049	90.00	1387.891185	24.059092	158.408404
	250	1337.798049	100.00	1337.798049	13.841454	0.000000
Pop. Size	**Max Itr.**	**Best Value**	**Succ. Rate (%)**	**Mean**	**Ave. Run Time (s)**	**Std. Dev.**
10	50	1337.798049	60.00	1337.798049	25.874616	0.000000
	100	1337.798049	70.00	1337.798049	45.744945	0.000000
	150	1337.798049	90.00	1337.798049	52.677546	0.000000
	200	1337.798049	90.00	1337.798049	60.189863	0.000000
	250	1337.798049	90.00	1337.798049	74.892136	0.000000
	300	1337.798049	100.00	1337.798049	67.605098	0.000000
	350	1337.798049	90.00	1337.798049	86.402059	0.000000
	400	1337.798049	100.00	1337.798049	91.761594	0.000000
	450	1337.798049	100.00	1337.798049	70.978083	0.000000
	500	1337.798049	100.00	1337.798049	86.532385	0.000000

The table above shows that for a small population size of 5, the population convergence rate for most of the runs are from 90% to 100%. This means that 9 to 10 out of 10 trials, the population converged to the optimal solution, $v_0 \rightarrow v_2 \rightarrow v_3 \rightarrow v_0 \rightarrow v_1 \rightarrow v_0$. However, the standard deviation reveals that in some cases, the best values obtained in all 10 trials were not the optimal solution. In these trials, the population converged at the suboptimal solution, $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_0 \rightarrow v_3 \rightarrow v_0$. This is reflected on the complete results shown on table D.1. We observe that the shortest set of trials averaged to about 12.04 seconds and the longest set of trials averaged to about 24.06 seconds. The difference between the longest and shortest runs was almost double the running time.

When we doubled the population size, there were no convergences on suboptimal solutions as evidenced by the standard deviations. All successful trials converged on the optimal solution. The trials that failed showed that their populations were not able to converge within the maximum iterations however, the best solution in the last population was the optimal one. This is seen on the complete results for this test shown on table D.1. It is observed that when the population size was doubled, the average running time also increased, more than just double the average running time for a population size of 5. This is because we have more members in the population that needs to converge. This might require more iterations for the population to fully converge to a solution. We can also observe that, as seen on figures 5.1a and 5.1b, when the maximum iterations is gradually increased there is a polynomial increase in the average running time for a population size of 5 and a logarithmic increase in the average running time when the population size is doubled. We observe that the shortest set of trials averaged to about 45.74 seconds and the longest set of trials averaged to about 91.76 seconds or about one and a half minutes. The difference was also a little over double the running time. When we compare the shortest and longest average running times for different population sizes, we observe that when the population size was doubled, the average running time was increased to more or less 4 times longer running time.



(a) Pop Size = 5



(b) Pop Size = 10

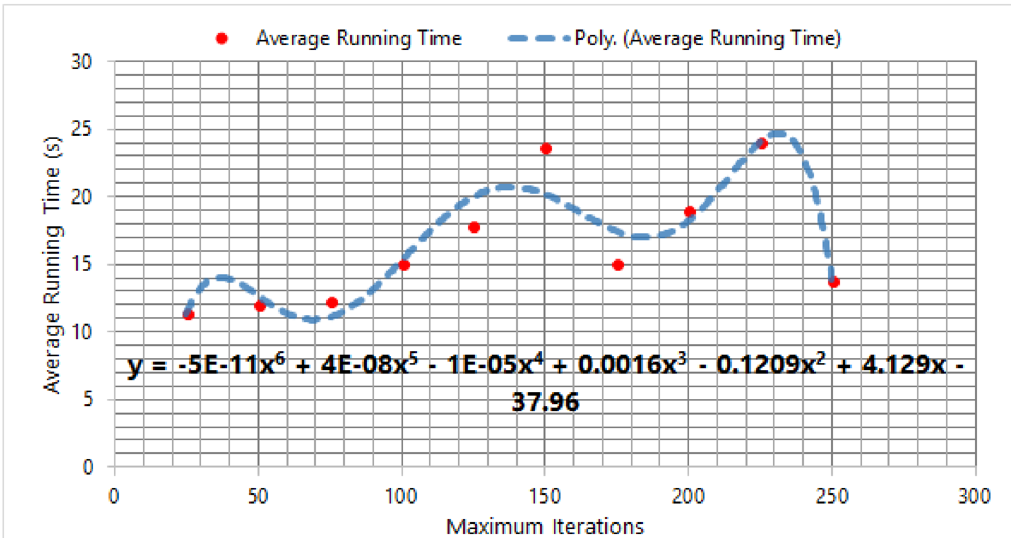
Figure 5.1: Ave. Run Time per Maximum Iterations 4 Nodes

Table 5.2: Results of the 5 Node Small Scale Test

Pop Size	**Max Itr.**	**Best Value**	**Succ. Rate (%)**	**Mean**	**Ave. Run Time (s)**	**Std. Dev.**
5	25	1317.218365	60.00	1437.839069	12.753919	301.943832
	50	1317.218365	50.00	1439.959649	15.112263	303.757647
	75	1317.218365	90.00	1317.886972	23.634884	2.114319
	100	1317.218365	70.00	1318.555578	30.000218	2.819092
	125	1317.218365	80.00	1329.917712	27.082303	37.867959
	150	1317.218365	90.00	1317.886972	31.221318	2.114319
	175	1317.218365	90.00	1317.886972	31.28952	2.114319
	200	1317.218365	80.00	1318.555578	26.880129	2.819092
	225	1317.218365	70.00	1319.224184	29.529753	3.229676
	250	1317.218365	90.00	1317.886972	29.548035	2.114319
Pop Size	**Max Itr.**	**Best Value**	**Succ. Rate (%)**	**Mean**	**Ave. Run Time (s)**	**Std. Dev.**
10	50	1317.218365	60.00	1317.218365	28.114193	0.000000
	100	1317.218365	80.00	1318.555578	42.935624	2.114319
	150	1317.218365	100.00	1317.218365	57.222174	0.000000
	200	1317.218365	90.00	1317.218365	65.603184	0.000000
	250	1317.218365	100.00	1317.218365	59.020941	0.000000
	300	1317.218365	100.00	1317.218365	73.810322	0.000000
	350	1317.218365	90.00	1317.218365	114.572148	0.000000
	400	1317.218365	70.00	1317.886972	116.272363	2.114319
	450	1317.218365	90.00	1317.886972	162.401904	2.114319
	500	1317.218365	90.00	1317.886972	120.888697	2.114319

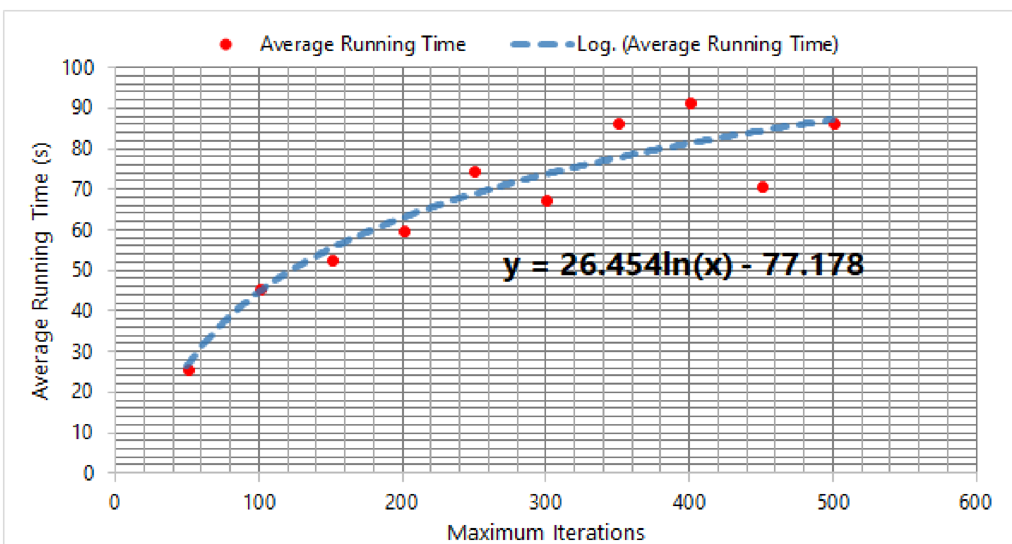
The results in the table above shows a similar trend from the smaller test case. For a population size of 5, most of the trials were able to converge and obtain the optimal solution, $v_0 \to v_1 \to v_2 \to v_0 \to v_3 \to v_4 \to v_0$. However, in some of the ten trials only a suboptimal solution was obtained as seen on the complete results on table E.1. Most of time, the failed trials converged to the route $v_0 \to v_1 \to v_4 \to v_0 \to v_2 \to v_3 \to v_0$. This is reflected by the mean and standard deviations. The mean, average values of results obtained, has a general trend of moving closer to the value of the optimal solution as the maximum iterations gradually increased from 50 to 250. This means that as the maximum iterations is gradually increased, the populations at each trial tended to converge more towards the optimal solution. As for the standard deviation, we see that on some tests, the value was high indicating that the results of the ten trials are far apart from each other. When the standard deviation is low, this means that the values of the ten trials are closer to each other. We also observe that the shortest set of trials averaged to about 15.11 seconds and the longest set of trials averaged to about 30.00 seconds. The difference was almost double the running time which was also observed in the smaller test case.

When we doubled the population size, the populations at each trial became more stable in converging at the optimal solution. We can see that there is a low standard deviation which indicates that in most the ten trials, the populations at each trial tended towards the optimal solution. The results obtained were less varied. When we observe the mean values, the average value of the ten trials in each test is close to or at the value of the optimal solution. In trials where the standard deviation is zero and the mean is equal to the optimal value, this means that all ten trials succeeded in obtaining the optimal solution. The running time shows the same trend as the previous test case. The average running time increases as the maximum iterations is gradually increased. For both population sizes, the convergence rate became more unstable compared to the previous test case. We can also observe that, as seen on figures 5.2a and 5.2b, when the maximum iterations is gradually increased there is a polynomial increase in the average running time for a population size of 5 and a logarithmic increase in average running time when the population size is doubled. This is similar to the previous results of the test for 4 nodes. We also observe that the shortest set of trials averaged to about 42.94 seconds and the longest set of trials averaged to about 162.40 seconds. The difference was almost quadruple the running time. We can see that there is a larger disparity in average running times for a population size of 10 as compared to 5. When we compare the shortest and longest average running times for different population sizes, we observe that when the population size was doubled, the average running time was increased to more or less 3 to 5 times longer.



x

(a) Pop Size = 5



(b) Pop Size = 10

Figure 5.2: Ave. Run Time per Maximum Iterations 4 Nodes

Table 5.3: Summary of Results for slightly larger problem sizes

# of Nodes	Pop Size	Max Itr	Best Feasible	Worst Value	Mean	Std. Dev.	Ave. Run Time (s)	Feasible Solutions
8	100	1000	2744.350205	3726.167111	3491.060733	388.661959	1563.182141	10/10
11	100	1000	6095.846685	7484.524339	5848.620502	988.962668	1562.796931	1/10
	250	2500	4232.927949	7086.245703	5705.185533	1024.513635	4933.202838	10/10
16	100	1000	N/A	9542.784453	8235.416304	717.4302625	1902.592442	0/10
	250	2500	8255.870347	9320.888833	8385.843161	755.65534	5699.270684	3/10

The table above shows that for 8 nodes, the algorithm worked and obtained the feasible solution, $v_0 \to v_2 \to v_4 \to v_7 \to v_0 \to v_5 \to v_6 \to v_0 \to v_1 \to v_0 \to v_3 \to v_0$. However, the values obtained at each trial was not consistent. This means that the initial population at each trial were varied and that the populations at each trial were also exploring different subsets of the solution space. This is indicated by the high standard deviation and a very large difference between the best obtained value and the mean of values obtained, which is $3,491.060733 - 2744.350205 = 746.710528$ pesos. This is also seen in the complete results in table F.1. The average running time for all ten trials, is a little to about 26 minutes.

As for the results for 11 nodes, the algorithm did not work well for a population size of 100 however, it was able to converge at one feasible solution which is obviously not the most optimal solution. When we increased the population size to 250, all ten trials converged. The best solution obtained was the route, $v_0 \to v_7 \to v_10 \to v_2 \to v_0 \to v_6 \to v_5 \to v_0 \to v_8 \to v_0 \to v_3 \to v_0 \to v_1 \to v_0 \to v_9 \to v_4 \to v_0$. The standard deviation is large which indicates that the solution values obtained are very far apart. This is evident of the large difference between the best and worst values found. When population size was increased 25 times, the average running time also increased by more than 3 times.

The results for 16 nodes showed terrible results. When the population size was 100, there were no feasible solutions found. When the population size was increased to 250, there were 3 out of 10 trials that were able to obtain feasible solutions. The best route found was $v_0 \to v_7 \to v_0 \to v_8 \to v_0 \to v_10 \to v_14 \to v_0 \to v_9 \to v_5 \to v_0 \to v_1 \to v_0 \to v_3 \to v_0 \to v_4 \to v_0 \to v_6 \to v_0 \to v_2 \to v_12 \to v_0 \to v_13 \to v_0 \to v_11 \to v_15 \to v_0$.

We find that when the population size was increased, the algorithm was able to obtain more feasible solutions. This may be because there are more members in the population that can explore the solution space.

Table 5.4: Summary of Results for 130 Nodes

Pop Size	**Max Itr**	**Best Value**	**Mean**	**Std. Dev**	**Ave. Run Time**	**Feasible**
100	1000	62488.740771	65283.288735	1500.374985	8032.586189	0/10
250	2500	62472.940313	64092.766331	1714.411220	15219.926172	0/10

The table above shows the results for the full scale problem however there were no feasible solutions found. This may be because the problem size was too large, hence, the PSO-GA algorithm might not be effective for large scale VRP problems. The number of possible solutions to the VRP problem increases as the number of nodes is increased hence, the complexity of the problem increases as well. This is because the number of possible combinations increases as we consider more nodes. Not only that but the number of vehicles needed also changes based on the instance of the demands generated.

Chapter 6

Conclusion and Recommendation

A model for the Baguio City waste collection vehicle routing problem was constructed which aimed to minimize the operational costs involved. The model considers the toal distance traveled by all vehicle and number of vehicles used. These were converted to cost in terms of Philippines Pesos (Php) by obtaining the fuel efficiency, fuel costs and driver wages. We looked into using the hybrid PSO-GA algorithm by Harish Garg[14] in order to solve the waste collection vehicle routing problem. This algorithm was previously used for constrained optimization problems on engineering designs. The results obtained during the preliminary testing show that the hybrid PSO-GA proposed by Harish Garg can indeed solve small scale vehicle routing problems however, in order to have a high success rate, the PSO population size must be large enough with a large enough maximum iterations. The routes obtained by the algorithm consists of sub-routes taken by each vehicle in the fleet wherein the barangays which had small demands (about 2 to 8 cubic meters) were grouped together while those with large demands (about 9 and above) were placed in its own sub-route.

The best set of vehicle routes for the full scale of the Baguio CVRP remains inconclusive since the algorithm was not able to obtain any feasible solutions. This is because the complexity of the problem has increased to a point where it cannot be handled by the parameters used or the algorithm itself. The solution space has expanded to a very large degree wherein the members of the population were unable to solve the problem. However, if there were enough time to test larger population sizes (i.e. 10,000 members), we might be able to find a feasible solution but the running time would be excessive.

For the next studies, it is recommend that the specific collection sites per barangay be used instead of barangay halls and landmarks to have a more accurate representation 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 and/or using a GIS system for real-time data. Demands can also be set higher than the capacity of a vehicle and partial collection can be employed. A simulation may instead be used in order to develop a more efficient schedule as in the studies of Son[35] and Akhtar et.al.[1].

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. As for the Baguio City full size problem, it is recommended to try a more appropriate algorithm that is directed towards solving large scale VRP problems such as the adaptive large neighborhood search heuristic by Buhrkal et.al.[2] and the guided variable neighborhood thresholding meta heuristic by Nuortio et.al.[28]. Both studies are discussed in the related literature.

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] National Solid Waste Management Commission, *National solid waste management status report 2008-2014*.

[27] ——, *Nswmc on-line database*.

[28] 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.

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

[30] 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).

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

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

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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
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)	, 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	, Baguio City, Benguet, Baguio, Benguet
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
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
Gabriela Silang	Gabriela Silang Covered Court, Gabriela Silang Road, Baguio City, Benguet, Philippines
General Emilio F. Aguinaldo (QuirinoMagsaysay, Lower)	Danes Bakeshop, Everlasting Street, Baguio

Table A.1 continued from previous page

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
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
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
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
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

Table A.1 continued from previous page

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
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
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
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
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	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
v_0	0	5.005	6.154	5.88	7.643	4.407	12.017	3.42	12.805	6.994	7.027	7.948	4.832
v_1	5.005	0	1.643	2.426	3.232	1.396	7.752	5.458	8.54	2.418	2.721	3.282	1.026
v_2	6.154	1.643	0	0.89	1.589	2.663	8.267	6.489	9.055	0.775	1.079	1.64	2.29
v_3	5.88	2.426	0.89	0	1.943	2.13	9.157	6.615	9.945	1.057	1.327	1.491	2.35
v_4	7.643	3.232	1.589	1.943	0	3.989	9.856	8.474	10.644	0.888	0.632	1.898	4.209
v_5	4.407	1.396	2.663	2.13	3.989	0	8.706	4.975	9.423	3.094	3.295	4.216	1.338
v_6	12.017	7.752	8.267	9.157	9.856	8.706	0	12.872	0.789	9.042	9.345	9.401	8.725
v_7	3.42	5.458	6.489	6.615	8.474	4.975	12.872	0	13.66	7.395	7.596	8.517	5.401
v_8	12.805	8.54	9.055	9.945	10.644	9.423	0.789	13.66	0	9.582	9.886	10.189	9.053
v_9	6.994	2.418	0.775	1.057	0.888	3.094	9.042	7.395	9.582	0	0.378	1.083	3.297
v_{10}	7.027	2.721	1.079	1.327	0.632	3.295	9.345	7.596	9.886	0.378	0	1.387	3.593
v_{11}	7.948	3.282	1.64	1.491	1.898	4.216	9.401	8.517	10.189	1.083	1.387	0	2.549
v_{12}	4.832	1.026	2.29	2.35	4.209	1.338	8.725	5.401	9.053	3.297	3.593	2.549	0
v_{13}	6.044	3.429	4.608	5.334	6.197	4.108	7.292	5.43	8.726	5.383	5.686	6.477	3.738
v_{14}	7.865	3.431	4.551	5.277	6.14	4.265	5.323	7.251	6.11	5.488	5.791	6.479	3.74
v_{15}	7.197	2.122	2.571	3.461	4.16	2.908	5.807	8.219	6.615	3.346	3.328	3.093	3.281
v_{16}	6.47	2.486	3.726	4.256	5.315	3.226	5.592	7.137	6.38	4.603	4.804	5.594	2.856
v_{17}	7.774	3.363	1.72	2.17	0.173	4.042	9.987	8.343	10.528	1.019	0.763	2.029	4.34
v_{18}	7.423	3.013	1.37	1.723	0.219	3.691	9.389	7.992	10.178	0.669	0.413	1.679	3.989
v_{19}	6.946	3.492	2.29	1.246	1.252	3.214	10.558	7.778	11.346	1.59	1.301	2.6	3.513
v_{20}	6.197	1.932	3.038	3.898	4.627	2.886	6.182	7.452	7.139	3.858	4.116	4.849	2.349
v_{21}	7.048	2.129	0.486	1.136	1.619	3.074	8.765	7.665	9.307	0.939	1.243	1.048	3.422
v_{22}	7.784	3.302	1.66	2.016	0.459	4.052	9.926	8.353	10.467	0.959	0.701	1.968	4.255
v_{23}	7.172	2.895	2.416	2.955	2.884	3.678	8.026	10.35	8.867	3.52	2.373	3.605	3.655
v_{24}	5.676	2.222	2.122	0.808	2.667	1.944	9.641	6.245	9.676	1.54	2.051	2.662	2.263
v_{25}	9.024	4.759	5.865	6.672	7.454	5.713	2.993	9.88	3.781	6.781	7.084	7.772	5.272
v_{26}	6.587	2.322	3.428	4.288	5.017	3.276	5.43	7.842	6.218	4.344	4.647	4.562	2.836
v_{27}	4.383	0.577	1.841	1.901	3.76	0.889	7.818	4.952	8.604	2.313	2.616	3.177	0.449
v_{28}	4.019	0.888	2.135	1.861	3.724	0.848	8.181	4.588	8.898	2.807	3.008	3.269	0.813
v_{29}	4.852	0.786	2.05	2.752	3.639	1.74	8.323	4.657	8.52	3.27	3.573	4.134	1.665
v_{30}	4.746	0.68	1.944	2.646	3.533	1.634	7.633	5.314	8.421	2.646	3.022	3.51	1.04
v_{31}	10.157	5.892	4.38	4.919	6.235	6.162	8.383	11.335	9.225	5.421	4.337	4.102	5.104
v_{32}	4.843	1.374	2.059	1.336	3.195	0.754	8.684	5.411	9.472	2.3	2.501	3.422	1.316
v_{33}	8.233	3.968	4.796	5.686	6.385	4.922	4.746	9.488	5.535	4.88	5.874	5.894	4.467
v_{34}	8.07	3.805	4.723	5.613	6.312	4.759	4.17	9.325	4.959	5.249	5.801	5.318	4.328
v_{35}	5.414	1.96	1.894	0.546	2.405	1.682	9.583	5.983	9.448	1.482	1.683	2.604	2.205
v_{36}	3.832	1.964	3.228	3.233	5.092	1.593	9.105	4.4	9.698	4.012	4.213	5.134	2.018
v_{37}	6.616	5.917	7.097	7.823	8.686	7.653	9.587	6.002	10.375	7.872	8.175	8.965	6.227
v_{38}	7.389	2.21	2.763	3.653	4.352	3.841	5.687	8.409	6.505	3.176	3.48	3.245	3.471
v_{39}	6.188	2.406	1.982	2.872	3.571	2.694	6.214	8.451	6.9	2.736	3.04	2.805	3.513
v_{40}	4.004	1.626	2.893	2.321	4.18	0.548	8.937	4.742	9.653	3.325	3.526	4.447	1.569
v_{41}	5.774	2.202	3.442	4.168	5.133	3.156	6.823	6.678	7.611	4.319	4.622	5.31	2.342
v_{42}	15.446	11.181	11.696	12.586	13.285	12.135	3.43	16.3	3.679	12.223	12.527	12.292	11.694

v_{43}	8.148	3.883	4.711	5.601	6.3	4.837	5.049	9.335	5.821	4.795	5.098	5.845	4.382
v_{44}	4.629	0.837	2.101	2.803	3.69	1.791	7.542	4.921	8.279	2.803	3.106	3.667	1.197
v_{45}	5.47	0.884	0.978	1.868	2.567	1.976	7.355	6.039	7.776	1.753	2.056	2.617	1.823
v_{46}	5.741	1.252	1.281	2.171	2.87	2.561	6.986	6.623	7.774	2.056	2.359	2.84	2.191
v_{47}	8.588	4.122	3.941	5.017	4.409	5.431	8.88	11.832	9.807	3.594	3.898	3.663	5.061
v_{48}	7.652	2.89	3.479	4.369	5.068	4.341	4.999	8.907	5.798	4.254	4.281	4.046	3.901
v_{49}	5.148	2.56	2.406	1.626	3.485	1.058	9.853	6.996	10.163	2.844	3.045	3.966	2.485
v_{50}	5.032	2.115	2.78	1.982	3.841	0.943	9.408	5.6	10.196	2.946	3.147	4.068	2.04
v_{51}	11.991	7.121	6.48	7.37	6.682	7.996	10.217	13.169	11.005	5.867	6.171	7.614	8.216
v_{52}	5.556	2.125	1.492	0.368	2.227	1.824	9.418	6.315	9.556	1.658	1.611	1.749	2.05
v_{53}	5.128	0.626	1.394	2.302	2.983	1.935	7.083	5.697	8.144	2.132	2.435	2.996	1.565
v_{54}	7.577	3.262	4.14	5.03	5.729	4.266	4.477	8.764	5.25	4.224	4.527	5.274	3.811
v_{55}	6.817	2.248	1.21	2.101	2.52	3.809	8.044	7.386	8.833	1.705	2.009	1.774	3.187
v_{56}	6.273	1.686	1.301	2.191	2.89	2.779	7.65	7.057	8.438	2.076	2.379	2.94	2.625
v_{57}	5.757	1.791	1.548	2.438	3.137	3.1	6.546	8.729	7.579	2.868	3.172	2.937	2.73
v_{58}	6.736	2.738	3.917	4.643	5.609	3.631	7.247	6.121	8.035	4.795	4.995	5.786	3.047
v_{59}	7.884	2.928	1.577	2.115	2.045	4.152	9.844	8.619	8.358	1.23	1.534	1.299	4.354
v_{60}	1.424	5.698	6.962	6.855	8.714	5.215	12.63	4.228	13.418	7.634	8.098	8.235	6.893
v_{61}	5.423	0.62	1.383	2.273	2.972	1.929	7.64	5.991	8.428	2.158	2.461	3.022	1.559
v_{62}	5.475	0.672	0.679	1.569	3.41	1.984	7.685	6.043	8.473	2.593	2.794	3.35	1.611
v_{63}	4.567	0.779	1.792	2.103	3.381	1.073	8.072	5.136	8.74	2.567	2.87	3.431	0.703
v_{64}	5.341	0.508	1.343	2.264	4.078	1.847	7.521	5.909	8.922	2.118	3.462	2.982	0.201
v_{65}	13.472	9.357	9.673	10.563	11.259	10.285	1.651	14.521	1.9	10.444	10.748	10.513	9.915
v_{66}	5.376	1.022	2.206	3.03	3.795	2	7.101	5.944	7.889	2.981	3.284	3.845	1.63
v_{67}	10.508	6.82	7.136	8.026	8.722	7.321	1.889	11.557	2.678	7.907	8.211	7.976	6.951
v_{68}	10.178	6.073	7.859	8.021	9.445	6.991	1.644	11.227	2.432	8.63	8.934	8.699	6.621
v_{69}	7.185	2.524	1.174	1.318	0.727	3.268	9.44	7.753	10.228	0.473	0.095	1.482	3.488
v_{70}	3.773	1.551	2.735	3.007	4.77	1.534	8.563	4.341	9.351	3.51	4.154	4.374	1.905
v_{71}	3.768	1.676	2.86	3.002	4.765	1.529	8.398	4.336	9.186	3.948	4.149	5.07	2.03
v_{72}	3.548	1.809	2.944	2.782	4.545	1.309	8.821	4.116	9.609	3.728	3.929	4.85	1.734
v_{73}	11.076	4.454	4.273	4.812	4.741	5.763	9.52	12.472	10.308	3.926	4.23	3.995	5.393
v_{74}	11.76	7.108	5.971	6.51	6.439	8.573	10.204	13.156	10.992	5.624	5.928	5.693	8.203
v_{75}	6.049	1.682	1.708	2.58	3.297	2.132	8.695	6.617	9.483	2.483	2.786	3.347	2.352
v_{76}	5.639	1.164	1.19	1.127	2.89	1.722	8.177	6.207	8.965	2.073	2.274	2.829	1.942
v_{77}	5.98	2.358	2.685	2.514	4.277	2.063	9.651	6.548	10.439	3.46	3.661	4.324	2.283
v_{78}	5.325	0.522	0.985	2.626	3.325	1.831	7.535	5.893	8.323	2.511	2.814	3.375	1.461
v_{79}	8.467	3.343	1.992	2.531	2.46	4.55	10.259	9.035	11.047	1.645	1.949	1.714	4.77
v_{80}	5.854	2.232	2.559	2.388	4.151	1.937	9.525	6.422	10.313	3.334	3.535	4.198	2.157
v_{81}	3.76	2.796	3.809	3.769	5.532	1.91	9.982	4.986	10.77	4.715	4.916	5.837	2.721
v_{82}	5.993	1.887	3.169	3.836	4.758	2.806	5.65	7.389	6.439	3.944	4.247	4.396	2.436
v_{83}	11.192	4.878	4.697	5.236	5.165	6.187	9.636	12.588	10.424	4.35	4.654	4.419	5.817
v_{84}	7.432	2.49	1.14	1.496	0.986	3.515	9.406	8	10.194	0.439	0.475	0.477	3.735
v_{85}	6.742	1.939	0.588	1.126	1.158	3.146	8.855	7.31	9.643	0.344	0.648	1.209	2.878
v_{86}	4.037	1.815	2.999	3.271	5.034	1.798	8.827	4.605	9.615	3.774	4.418	4.638	2.169
v_{87}	5.795	1.518	0.489	1.379	2.078	2.786	8.059	6.364	8.848	1.254	1.567	2.118	0.784
v_{88}	10.757	6.492	4.407	5.297	5.996	5.374	9.036	12.012	9.825	5.182	5.485	5.541	4.865
v_{89}	7.639	5.525	3.178	4.068	4.767	4.145	8.069	11.045	8.858	3.953	4.256	4.312	3.636
v_{90}	5.015	1.561	1.38	1.656	2.969	1.283	8.145	5.584	8.934	2.634	2.458	2.514	0.885
v_{91}	5.676	1.399	1.161	2.051	2.75	2.182	7.926	6.245	8.715	1.936	2.239	2.295	0.666
v_{92}	3.689	0.626	1.89	2.592	3.479	1.58	7.645	4.945	8.433	2.665	2.968	3.024	2.63
v_{93}	5.39	2.094	3.334	4.06	4.923	3.048	6.664	7.106	7.452	4.109	4.412	4.468	3.792
v_{94}	5.808	3.042	2.976	1.628	3.487	2.764	9.741	7.065	10.53	2.575	2.871	3.008	2.481
v_{95}	5.941	3.305	3.422	2.642	4.501	2.311	10.544	7.367	11.332	3.589	3.885	4.022	3.397
v_{96}	3.734	2.953	4.032	3.252	5.111	1.708	9.961	5.16	10.749	4.199	4.495	4.632	3.053
v_{97}	6.446	2.869	3.975	4.835	5.564	3.823	5.139	8.389	5.927	4.75	5.053	5.109	4.873
v_{98}	8.665	7.239	4.892	5.782	6.481	5.859	9.783	12.759	10.572	5.667	5.97	6.026	5.35
v_{99}	3.308	2.063	3.469	2.897	4.756	1.124	9.373	4.734	10.09	3.844	4.14	4.277	2.974
v_{100}	3.274	2.564	3.969	3.397	5.256	1.624	9.995	4.999	10.783	4.728	4.929	5.85	2.734
v_{101}	6.186	3.42	3.32	2.006	3.865	3.142	10.839	7.726	11.627	2.738	2.939	3.86	3.461
v_{102}	5.35	2.584	2.484	1.17	3.029	2.306	9.249	6.89	10.791	1.902	2.103	3.024	2.625

v_{103}	5.628	2.862	2.762	1.448	3.307	2.584	9.527	6.885	10.316	2.395	2.691	2.828	2.267
v_{104}	6.482	3.716	3.616	2.302	4.161	3.438	10.381	7.739	11.17	3.249	3.545	3.682	3.121
v_{105}	4.518	1.414	2.678	3.38	4.267	2.368	7.494	5.774	8.282	3.453	3.756	3.812	3.418
v_{106}	4.145	0.354	1.395	2.32	2.984	1.308	7.658	5.401	8.447	2.17	2.473	2.529	2.358
v_{107}	3.658	1.08	2.344	3.046	3.933	2.065	7.774	4.914	8.562	3.119	3.422	3.478	3.084
v_{108}	3.307	1.431	2.695	3.397	4.284	1.714	8.125	4.563	8.913	3.47	3.773	3.829	3.435
v_{109}	3.567	1.172	2.436	3.138	4.025	1.974	7.778	4.823	8.566	3.592	3.895	3.57	3.176
v_{110}	6.84	3.251	3.067	3.957	4.656	4.034	8.712	10.706	9.501	3.842	4.145	4.201	3.525
v_{111}	8.336	4.759	5.865	6.725	7.454	5.713	2.993	9.88	3.781	6.64	6.943	6.999	6.763
v_{112}	7.133	3.679	1.549	1.433	0.51	3.401	9.568	7.702	10.357	0.848	0.592	0.941	2.459
v_{113}	3.881	2.676	3.94	3.833	5.692	2.193	9.608	2.81	10.396	4.78	5.076	5.213	3.871
v_{114}	2.791	6.817	8.081	7.974	9.833	6.334	13.749	5.347	14.537	8.921	9.217	9.354	8.012
v_{115}	6.674	3.097	4.203	5.063	5.792	4.051	5.443	7.653	6.231	4.978	5.281	5.337	5.101
v_{116}	6.475	3.709	2.506	1.463	1.468	3.431	10.525	7.732	11.314	1.805	1.518	1.898	3.114
v_{117}	5.393	2.627	1.161	0.381	1.96	2.349	9.183	6.65	9.972	1.328	1.344	1.761	2.032
v_{118}	6.78	3.203	4.031	4.921	5.62	4.157	4.749	8.723	5.538	4.806	5.109	5.165	4.489
v_{119}	6.074	3.009	4.249	4.975	5.838	3.963	7.578	6.149	8.366	5.024	5.327	5.383	4.707
v_{120}	5.844	5.773	7.013	7.739	8.602	7.57	6.962	5.919	7.75	7.788	8.091	8.147	7.471
v_{121}	10.158	10.088	11.328	12.054	12.917	11.884	13.818	10.233	14.606	12.103	12.406	12.462	11.786
v_{122}	7.471	3.894	4.587	5.477	6.176	4.848	3.854	9.414	4.643	5.362	5.665	5.721	5.045
v_{123}	6.049	1.662	1.53	2.42	3.119	2.447	6.773	8.086	7.562	2.305	2.608	2.664	1.988
v_{124}	4.829	2.063	1.37	1.067	2.926	1.785	8.135	6.086	8.924	2.014	2.31	2.504	0.875
v_{125}	7.3	3.435	4.675	5.401	6.264	4.389	5.678	7.375	6.466	5.45	5.753	5.809	5.133
v_{126}	7.78	4.203	4.136	5.026	5.725	4.473	6.747	9.723	7.536	4.911	5.214	5.27	4.594
v_{127}	4.983	1.394	0.476	1.366	2.065	2.662	8.046	6.24	8.835	1.251	1.554	1.61	0.66
v_{128}	5.577	2.811	0.603	0.496	1.561	2.533	8.625	6.834	9.414	0.674	1.051	1.107	1.513
v_{129}	2.819	2.038	3.302	3.084	4.943	1.311	9.046	4.245	9.834	4.031	4.327	4.464	3.161
NODES	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}
v_0	6.044	7.865	7.197	6.47	7.774	7.423	6.946	6.197	7.048	7.784	7.172	5.676	9.024
v_1	3.429	3.431	2.122	2.486	3.363	3.013	3.492	1.932	2.129	3.302	2.895	2.222	4.759
v_2	4.608	4.551	2.571	3.726	1.72	1.37	2.29	3.038	0.486	1.66	2.416	2.122	5.865
v_3	5.334	5.277	3.461	4.256	2.17	1.723	1.246	3.898	1.136	2.016	2.955	0.808	6.672
v_4	6.197	6.14	4.16	5.315	0.173	0.219	1.252	4.627	1.619	0.459	2.884	2.667	7.454
v_5	4.108	4.265	2.908	3.226	4.042	3.691	3.214	2.886	3.074	4.052	3.678	1.944	5.713
v_6	7.292	5.323	5.807	5.592	9.987	9.389	10.558	6.182	8.765	9.926	8.026	9.641	2.993
v_7	5.43	7.251	8.219	7.137	8.343	7.992	7.778	7.452	7.665	8.353	10.35	6.245	9.88
v_8	8.726	6.11	6.615	6.38	10.528	10.178	11.346	7.139	9.307	10.467	8.867	9.676	3.781
v_9	5.383	5.488	3.346	4.603	1.019	0.669	1.59	3.858	0.939	0.959	3.52	1.54	6.781
v_{10}	5.686	5.791	3.328	4.804	0.763	0.413	1.301	4.116	1.243	0.701	2.373	2.051	7.084
v_{11}	6.477	6.479	3.093	5.594	2.029	1.679	2.6	4.849	1.048	1.968	3.605	2.662	7.772
v_{12}	3.738	3.74	3.281	2.856	4.34	3.989	3.513	2.349	3.422	4.255	3.655	2.263	5.272
v_{13}	0	1.969	3.264	1.854	6.431	6.081	6.204	2.169	5.234	6.37	5.068	4.934	4.299
v_{14}	1.969	0	2.88	1.797	6.271	5.921	6.44	2.171	5.236	6.21	4.542	5.19	2.329
v_{15}	3.264	2.88	0	2.381	4.291	3.941	4.862	1.636	3.255	4.23	2.002	3.945	4.559
v_{16}	1.854	1.797	2.381	0	5.548	5.198	5.322	1.139	4.41	5.487	5.044	4.052	2.599
v_{17}	6.431	6.271	4.291	5.548	0	0.35	1.383	4.758	1.899	0.59	3.015	2.798	7.585
v_{18}	6.081	5.921	3.941	5.198	0.35	0	1.032	4.408	1.549	0.528	4.115	2.137	7.235
v_{19}	6.204	6.44	4.862	5.322	1.383	1.032	0	5.061	2.39	1.561	3.586	1.971	7.888
v_{20}	2.169	2.171	1.636	1.139	4.758	4.408	5.061	0	3.833	4.697	3.965	3.545	3.189
v_{21}	5.234	5.236	3.255	4.41	1.899	1.549	2.39	3.833	0	1.714	1.843	1.809	6.438
v_{22}	6.37	6.21	4.23	5.487	0.59	0.528	1.561	4.697	1.714	0	4.404	2.498	7.665
v_{23}	5.068	4.542	2.002	5.044	3.015	4.115	3.586	3.965	1.843	4.404	0	4.415	5.856
v_{24}	4.934	5.19	3.945	4.052	2.798	2.137	1.971	3.545	1.809	2.498	4.415	0	6.468
v_{25}	4.299	2.329	4.559	2.599	7.585	7.235	7.888	3.189	6.438	7.665	5.856	6.468	0
v_{26}	2.56	2.451	1.757	0.924	5.148	4.798	5.451	0.753	4.001	5.228	3.419	4.032	2.437
v_{27}	3.289	3.377	2.02	2.407	3.891	3.54	3.064	1.998	2.902	3.197	2.79	1.971	4.823
v_{28}	3.583	3.74	2.383	2.701	3.855	3.505	3.651	2.194	2.708	3.794	3.153	1.657	5.117
v_{29}	2.823	3.719	3.503	2.103	3.77	3.42	3.915	1.816	2.623	3.709	4.415	2.861	4.739
v_{30}	2.717	2.66	2.813	1.997	3.664	3.314	3.809	1.71	2.517	3.53	4.309	2.236	4.633

v_{31}	6.13	5.997	3.407	5.171	6.366	6.016	6.937	4.323	5.219	4.918	2.344	5.616	6.891
v_{32}	4.086	4.243	2.886	3.418	3.248	2.975	2.499	2.697	2.337	3.258	4.352	1.15	5.62
v_{33}	4.206	4.082	2.858	3.478	5.825	6.166	7.087	2.399	5.369	5.764	5.058	5.663	5.933
v_{34}	4.043	3.943	2.282	3.315	6.443	6.093	7.014	2.236	5.296	6.134	4.534	5.343	5.357
v_{35}	4.672	5.132	3.887	3.79	2.43	2.079	1.709	3.283	1.547	2.44	4.187	0.332	6.206
v_{36}	4.001	4.018	4.285	3.281	5.223	4.872	4.132	2.994	3.801	4.97	5.593	2.862	5.917
v_{37}	2.637	4.458	5.752	4.343	8.919	8.467	8.986	4.511	7.67	8.858	7.088	7.423	6.595
v_{38}	3.456	3.07	0.152	2.573	4.122	4.133	5.054	1.828	2.95	4.422	2.194	3.391	4.196
v_{39}	3.548	3.112	0.485	2.287	3.702	3.332	4.273	1.92	2.555	3.621	1.413	3.635	4.426
v_{40}	4.338	4.496	3.139	3.456	4.311	3.96	3.484	2.949	3.322	4.283	4.255	2.175	5.872
v_{41}	1.408	1.351	2.097	0.688	13.169	4.812	5.331	1.003	4.015	5.203	3.433	4.081	3.83
v_{42}	10.721	8.751	9.256	9.021	13.4	13.066	13.739	9.611	11.997	13.355	11.455	12.317	6.422
v_{43}	4.121	3.997	2.227	3.171	6.431	6.081	7.002	2.314	5.284	6.37	4.47	5.578	4.906
v_{44}	2.542	2.503	2.49	1.84	3.821	3.471	3.966	1.722	2.601	3.687	4.152	2.393	4.498
v_{45}	3.737	3.577	1.659	2.752	2.698	2.348	3.268	2.109	1.551	2.637	1.8	2.648	5.032
v_{46}	3.85	3.852	1.29	2.967	3.001	2.651	3.572	2.317	1.854	2.94	1.431	2.655	5.144
v_{47}	6.712	6.494	3.904	5.668	5.847	5.497	6.418	4.905	4.7	4.479	2.264	5.177	7.497
v_{48}	3.625	3.516	1.019	2.897	5.199	4.849	5.77	1.818	4.052	4.862	3.262	4.853	4.41
v_{49}	4.848	5.412	4.055	4.587	3.616	3.265	2.789	4.033	2.627	3.802	5.169	1.539	6.382
v_{50}	4.733	4.967	3.61	4.142	3.972	3.621	3.066	3.344	3.001	3.904	5.076	1.796	6.267
v_{51}	7.964	7.831	5.241	7.005	8.2	7.85	8.771	6.301	7.053	6.752	4.178	7.854	8.725
v_{52}	4.814	4.977	3.276	4.152	2.358	2.007	1.531	3.598	1.713	2.616	3.556	0.27	6.348
v_{53}	3.078	3.318	1.387	2.493	3.114	2.764	3.465	1.45	1.967	3.053	3.331	2.215	4.373
v_{54}	3.55	3.551	1.656	2.6	5.169	5.51	6.431	1.743	4.713	5.108	3.899	5.007	4.335
v_{55}	5.659	5.448	3.468	4.623	2.651	2.301	3.222	4.031	1.479	2.59	2.787	2.585	6.954
v_{56}	4.416	4.494	1.954	3.669	3.021	2.671	3.591	2.788	1.874	2.96	2.128	2.675	5.711
v_{57}	3.54	3.39	0.85	2.565	3.268	3.464	3.839	1.877	2.642	3.753	1.978	2.922	4.835
v_{58}	0.839	1.278	2.573	1.163	5.637	5.287	5.806	1.331	4.593	5.679	4.377	4.243	3.608
v_{59}	6.185	6.128	1.8	5.303	2.176	1.826	2.746	3.556	1.004	2.115	2.312	2.812	6.479
v_{60}	6.851	8.64	7.651	7.001	8.845	8.494	8.018	6.714	7.856	8.76	8.327	6.768	9.637
v_{61}	3.919	3.862	2.004	3.037	3.103	2.673	3.673	2.349	1.876	3.042	2.085	3.053	5.176
v_{62}	3.976	3.919	1.989	3.094	3.541	3.19	2.713	2.406	2.197	3.37	2.413	1.443	5.233
v_{63}	3.688	3.631	2.274	2.806	3.512	3.162	3.169	2.036	2.278	3.451	3.044	2.276	5.079
v_{64}	3.812	3.755	1.825	2.725	3.063	2.713	3.33	2.242	1.941	3.002	2.595	2.06	5.069
v_{65}	9.588	6.972	7.477	7.242	11.39	11.04	11.96	8.001	10.194	11.576	9.729	10.538	4.643
v_{66}	2.321	2.264	2.281	1.439	3.926	3.576	4.096	1.281	2.704	3.865	4.085	2.826	4.108
v_{67}	6.624	4.008	4.94	4.278	8.853	8.503	9.423	5.037	7.756	8.792	7.192	8.001	1.679
v_{68}	6.294	3.678	5.663	3.948	9.576	9.226	9.087	4.707	8.355	9.515	7.915	7.817	1.349
v_{69}	5.781	5.724	3.744	4.899	0.858	0.508	1.196	4.211	1.179	0.796	2.468	1.946	7.038
v_{70}	3.846	3.959	3.743	3.134	4.901	4.55	4.073	2.743	3.251	4.394	5.547	2.803	5.57
v_{71}	3.398	3.359	3.673	2.697	4.896	4.545	4.068	2.578	3.946	4.906	5.477	2.798	5.405
v_{72}	3.748	3.709	4.023	3.047	4.676	4.325	3.848	3.001	3.854	4.686	4.074	2.578	5.828
v_{73}	7.191	7.134	4.544	6.308	4.872	4.522	5.442	5.604	3.629	4.811	2.586	5.509	8.028
v_{74}	7.875	7.818	5.228	6.992	6.57	6.22	7.14	6.288	5.448	6.509	4.165	7.207	8.712
v_{75}	4.986	4.929	2.999	4.104	3.428	3.078	3.646	3.416	2.231	3.367	3.423	2.376	6.727
v_{76}	4.468	4.411	2.481	3.586	3.021	2.67	2.193	2.898	1.874	2.849	2.905	0.923	5.725
v_{77}	5.267	5.21	3.853	4.385	4.408	4.057	3.58	3.831	3.369	4.344	4.4	2.31	6.658
v_{78}	3.826	3.769	1.839	2.944	3.456	3.106	4.026	2.256	2.267	3.395	2.34	2.657	5.083
v_{79}	6.6	6.543	4.563	5.718	2.591	2.241	3.161	5.03	1.461	2.53	0.516	3.228	7.857
v_{80}	5.141	5.084	3.727	4.259	4.282	3.931	3.454	3.705	3.309	4.218	4.274	2.184	6.532
v_{81}	5.136	5.097	4.291	4.435	5.663	5.312	4.835	4.162	4.946	5.673	5.061	3.565	6.989
v_{82}	2.108	2.051	1.36	1.225	4.889	4.539	4.902	0.521	3.7	4.828	3.612	3.632	2.945
v_{83}	7.307	7.25	4.66	6.424	5.296	4.946	5.866	5.72	4.166	5.235	3.01	5.933	8.144
v_{84}	5.747	5.69	3.71	4.865	1.117	0.767	1.687	4.177	1.224	1.056	2.434	2.193	7.004
v_{85}	5.196	5.139	3.159	4.314	1.289	0.939	1.859	3.626	0.544	1.229	2.196	1.824	6.453
v_{86}	4.11	4.223	4.007	3.398	5.165	4.814	4.337	3.007	3.537	4.658	5.811	3.067	5.834
v_{87}	4.272	4.274	2.29	3.389	2.209	1.859	2.78	2.644	1.017	2.148	2.782	1.863	5.892
v_{88}	6.73	6.731	3.965	6.002	6.127	5.777	6.698	4.923	3.97	6.066	2.544	5.781	7.515
v_{89}	5.763	5.764	2.151	5.035	4.898	4.548	5.469	3.956	3.957	4.837	1.315	4.552	6.548
v_{90}	4.273	4.275	2.459	3.391	3.1	2.75	2.819	2.884	1.925	3.039	3.673	1.569	5.807

v_{91}	4.153	4.155	2.24	3.27	2.881	2.531	3.452	2.525	1.706	2.82	3.454	2.535	5.448
v_{92}	3.098	3.1	2.661	2.216	3.61	3.26	3.755	1.729	2.463	3.549	3.255	2.505	4.652
v_{93}	1.824	1.826	1.663	0.942	5.054	4.704	5.223	0.748	3.907	4.993	3.325	3.973	3.671
v_{94}	5.754	5.756	4.055	4.872	3.618	3.267	2.791	4.365	2.629	3.533	5.269	1.299	7.288
v_{95}	6.017	6.019	5.56	5.135	4.632	4.281	3.805	4.628	3.643	4.547	6.185	2.555	7.551
v_{96}	5.052	5.054	4.982	4.332	5.242	4.891	4.415	4.045	4.253	5.157	5.582	3.165	6.968
v_{97}	3.107	2.82	2.304	0.709	5.695	5.345	5.998	1.299	4.548	5.634	3.966	4.748	2.146
v_{98}	7.477	7.478	3.865	6.749	6.612	6.262	7.183	5.67	5.465	6.551	3.029	6.266	8.262
v_{99}	4.596	4.598	4.318	3.893	4.887	4.536	4.06	3.386	3.898	4.802	4.692	2.81	6.309
v_{100}	5.149	5.11	4.304	4.448	5.676	5.325	4.848	4.175	4.783	5.686	5.074	3.578	7.002
v_{101}	6.445	6.388	5.143	5.563	3.686	3.335	2.858	5.009	2.793	3.696	4.636	1.284	7.836
v_{102}	5.609	5.552	4.307	4.727	2.85	2.499	2.022	4.173	1.957	2.86	3.8	0.448	7
v_{103}	5.574	5.576	3.841	4.692	3.438	3.087	2.611	4.451	2.235	3.138	4.078	0.726	7.278
v_{104}	6.428	6.43	4.695	5.546	4.292	3.941	3.465	5.303	3.087	3.99	4.93	1.578	8.13
v_{105}	2.131	2.133	2.494	1.411	4.398	4.048	4.543	1.726	3.178	4.264	4.624	2.97	4.553
v_{106}	3.381	3.383	1.972	2.498	3.115	2.765	3.483	1.842	2.178	3.264	2.857	2.131	4.669
v_{107}	2.843	2.845	2.773	2.123	4.064	3.714	4.209	1.858	2.917	4.003	4.435	2.959	4.781
v_{108}	3.194	3.196	3.124	2.474	4.415	4.065	4.56	2.209	3.268	4.354	4.786	3.31	5.132
v_{109}	2.935	2.937	2.778	2.215	4.156	3.806	4.301	1.95	3.009	4.095	4.527	3.051	4.873
v_{110}	5.424	5.426	2.04	4.541	4.787	4.437	5.358	3.796	3.64	4.726	1.204	4.441	7.191
v_{111}	4.299	2.33	4.194	2.599	7.585	7.235	7.888	3.189	6.438	7.524	5.856	6.638	0
v_{112}	6.26	6.262	3.799	5.377	0.641	0.29	0.742	4.632	1.604	0.819	4.294	1.847	7.555
v_{113}	4.915	4.701	4.629	3.979	5.823	5.472	4.996	3.692	4.834	5.738	5.305	3.746	6.615
v_{114}	7.97	9.759	8.77	8.12	9.964	9.613	9.137	7.833	8.975	9.879	9.446	7.887	10.756
v_{115}	2.37	2.372	2.532	0.517	5.923	5.573	6.226	1.527	4.776	5.862	4.194	4.976	2.45
v_{116}	6.421	6.423	4.756	5.539	1.599	1.248	0.217	5.032	2.561	1.777	5.251	1.877	7.955
v_{117}	5.339	5.341	3.414	4.457	2.091	1.74	1.264	3.95	1.382	2.269	3.906	0.795	6.873
v_{118}	3.441	3.442	1.571	1.604	5.751	5.401	6.322	1.634	4.604	5.69	3.79	5.082	3.117
v_{119}	0.867	1.808	2.578	1.495	5.969	5.619	6.138	1.662	4.822	5.908	4.24	4.888	4.585
v_{120}	2.553	3.671	5.342	4.259	8.733	8.383	8.902	4.427	7.586	8.672	7.004	7.652	3.97
v_{121}	6.868	8.656	9.657	8.574	13.048	12.698	13.217	8.742	11.901	12.987	11.319	11.967	10.826
v_{122}	4.132	4.133	2.127	3.404	6.307	5.957	6.878	2.325	5.16	6.246	4.346	5.961	5.041
v_{123}	2.804	2.806	1.087	1.921	3.25	2.9	3.821	1.176	2.103	3.189	2.277	2.904	4.099
v_{124}	4.362	4.364	2.449	3.479	3.057	2.706	2.23	2.734	1.943	3.029	3.663	0.98	5.657
v_{125}	2.093	0.356	3.004	1.921	6.395	6.045	6.564	2.089	5.248	6.334	4.666	5.314	2.685
v_{126}	4.441	4.442	1.676	3.713	5.856	5.506	6.427	2.634	4.709	5.795	1.331	5.51	5.226
v_{127}	4.148	4.15	2.277	3.265	2.196	1.846	2.767	2.52	1.049	2.135	2.769	1.85	5.443
v_{128}	5.314	5.316	2.856	4.431	1.692	1.342	1.728	3.686	0.825	1.632	3.348	0.979	6.609
v_{129}	4.137	4.139	4.067	3.417	5.074	4.723	4.247	3.13	4.085	4.989	4.667	2.997	6.053
NODES	v_{26}	v_{27}	v_{28}	v_{29}	v_{30}	v_{31}	v_{32}	v_{33}	v_{34}	v_{35}	v_{36}	v_{37}	v_{38}
v_0	6.587	4.383	4.019	4.852	4.746	10.157	4.843	8.233	8.07	5.414	3.832	6.616	7.389
v_1	2.322	0.577	0.888	0.786	0.68	5.892	1.374	3.968	3.805	1.96	1.964	5.917	2.21
v_2	3.428	1.841	2.135	2.05	1.944	4.38	2.059	4.796	4.723	1.894	3.228	7.097	2.763
v_3	4.288	1.901	1.861	2.752	2.646	4.919	1.336	5.686	5.613	0.546	3.233	7.823	3.653
v_4	5.017	3.76	3.724	3.639	3.533	6.235	3.195	6.385	6.312	2.405	5.092	8.686	4.352
v_5	3.276	0.889	0.848	1.74	1.634	6.162	0.754	4.922	4.759	1.682	1.593	7.653	3.841
v_6	5.43	7.818	8.181	8.323	7.633	8.383	8.684	4.746	4.17	9.583	9.105	9.587	5.687
v_7	7.842	4.952	4.588	4.657	5.314	11.335	5.411	9.488	9.325	5.983	4.4	6.002	8.409
v_8	6.218	8.604	8.898	8.52	8.421	9.225	9.472	5.535	4.959	9.448	9.698	10.375	6.505
v_9	4.344	2.313	2.807	3.27	2.646	5.421	2.3	4.88	5.249	1.482	4.012	7.872	3.176
v_{10}	4.647	2.616	3.008	3.573	3.022	4.337	2.501	5.874	5.801	1.683	4.213	8.175	3.48
v_{11}	4.562	3.177	3.269	4.134	3.51	4.102	3.422	5.894	5.318	2.604	5.134	8.965	3.245
v_{12}	2.836	0.449	0.813	1.665	1.04	5.104	1.316	4.467	4.328	2.205	2.018	6.227	3.471
v_{13}	2.56	3.289	3.583	2.823	2.717	6.13	4.086	4.206	4.043	4.672	4.001	2.637	3.456
v_{14}	2.451	3.377	3.74	3.719	2.66	5.997	4.243	4.082	3.943	5.132	4.018	4.458	3.07
v_{15}	1.757	2.02	2.383	3.503	2.813	3.407	2.886	2.858	2.282	3.887	4.285	5.752	0.152
v_{16}	0.924	2.407	2.701	2.103	1.997	5.171	3.418	3.478	3.315	3.79	3.281	4.343	2.573
v_{17}	5.148	3.891	3.855	3.77	3.664	6.366	3.248	5.825	6.443	2.43	5.223	8.919	4.122
v_{18}	4.798	3.54	3.505	3.42	3.314	6.016	2.975	6.166	6.093	2.079	4.872	8.467	4.133

v_{19}	5.451	3.064	3.651	3.915	3.809	6.937	2.499	7.087	7.014	1.709	4.132	8.986	5.054
v_{20}	0.753	1.998	2.194	1.816	1.71	4.323	2.697	2.399	2.236	3.283	2.994	4.511	1.828
v_{21}	4.001	2.902	2.708	2.623	2.517	5.219	2.337	5.369	5.296	1.547	3.801	7.67	2.95
v_{22}	5.228	3.197	3.794	3.709	3.53	4.918	3.258	5.764	6.134	2.44	4.97	8.858	4.422
v_{23}	3.419	2.79	3.153	4.415	4.309	2.344	4.352	5.058	4.534	4.187	5.593	7.088	2.194
v_{24}	4.032	1.971	1.657	2.861	2.236	5.616	1.15	5.663	5.343	0.332	2.862	7.423	3.391
v_{25}	2.437	4.823	5.117	4.739	4.633	6.891	5.62	5.933	5.357	6.206	5.917	6.595	4.196
v_{26}	0	2.388	2.751	2.893	2.203	4.454	3.254	2.539	2.4	4.143	3.675	5.048	1.758
v_{27}	2.388	0	0.364	1.216	0.591	5.933	0.867	4.018	3.879	1.756	1.569	5.778	3.022
v_{28}	2.751	0.364	0	1.462	0.885	6.227	1.284	4.312	4.173	2.343	1.205	6.072	3.316
v_{29}	2.893	1.216	1.462	0	0.106	5.849	1.718	3.934	3.795	2.607	1.738	5.312	2.943
v_{30}	2.203	0.591	0.885	0.106	0	5.743	1.612	3.828	3.689	2.501	1.632	5.206	2.837
v_{31}	4.454	5.933	6.227	5.849	5.743	0	6.14	5.468	4.892	5.962	7.245	8.618	3.245
v_{32}	3.254	0.867	1.284	1.718	1.612	6.14	0	4.815	4.676	1.191	2.029	6.575	3.819
v_{33}	2.539	4.018	4.312	3.934	3.828	5.468	4.815	0	0.576	5.789	5.321	6.694	2.216
v_{34}	2.4	3.879	4.173	3.795	3.689	4.892	4.676	0.576	0	5.626	5.158	6.531	2.143
v_{35}	4.143	1.756	2.343	2.607	2.501	5.962	1.191	5.789	5.626	0	2.6	7.161	3.163
v_{36}	3.675	1.569	1.205	1.738	1.632	7.245	2.029	5.321	5.158	2.6	0	7.077	4.121
v_{37}	5.048	5.778	6.072	5.312	5.206	8.618	6.575	6.694	6.531	7.161	7.077	0	5.616
v_{38}	1.758	3.022	3.316	2.943	2.837	3.245	3.819	2.216	2.143	3.163	4.121	5.616	0
v_{39}	1.989	3.064	3.358	2.985	2.879	3.772	3.861	2.743	2.67	3.407	4.163	5.658	0.637
v_{40}	3.507	1.12	0.913	1.971	1.865	6.393	0.985	5.153	4.99	1.913	1.553	7.42	3.331
v_{41}	1.393	1.893	2.187	1.408	1.302	4.963	2.69	3.039	2.876	3.276	2.586	3.886	2.289
v_{42}	8.859	11.245	11.539	11.161	11.055	11.866	12.042	8.176	7.6	12.089	12.339	13.016	9.146
v_{43}	2.454	3.933	4.227	3.849	3.743	4.837	4.73	1.241	1.281	5.316	5.027	6.543	2.117
v_{44}	2.112	0.748	1.042	0.263	0.157	5.682	1.545	3.758	3.595	2.131	1.441	5.031	3.009
v_{45}	2.454	1.374	1.668	1.583	1.477	3.735	2.585	3.884	3.811	2.42	2.761	6.123	1.851
v_{46}	2.707	1.742	2.036	1.951	1.845	3.366	2.888	3.515	3.442	2.723	3.129	6.376	1.482
v_{47}	4.951	4.612	4.906	6.346	6.24	2.345	5.705	5.912	5.336	4.915	7.524	9.04	3.794
v_{48}	1.973	3.452	3.746	3.368	3.262	3.62	4.249	1.394	1.231	3.843	4.546	6.062	0.9
v_{49}	4.423	2.036	2.079	2.887	2.781	7.309	1.329	6.069	5.906	1.432	2.824	9.674	4.247
v_{50}	3.978	1.591	1.473	2.442	2.336	6.864	0.723	5.624	5.461	1.534	2.218	8.278	3.802
v_{51}	6.288	7.767	8.061	7.683	7.577	1.977	8.564	7.249	6.673	7.188	8.861	10.377	5.131
v_{52}	3.988	1.601	2.188	2.452	2.346	5.635	1.036	5.634	5.471	0.246	2.933	7.523	3.468
v_{53}	2.195	1.116	1.41	1.621	1.515	4.641	1.913	3.612	3.539	1.846	2.799	5.864	1.579
v_{54}	1.883	3.362	3.656	3.278	3.172	4.266	4.159	0.656	0.709	4.745	4.456	5.972	1.546
v_{55}	4.325	2.738	3.032	2.947	2.841	5.543	2.956	5.693	5.62	2.791	4.125	7.994	3.66
v_{56}	3.371	2.176	2.47	2.385	2.279	4.029	2.908	4.179	4.106	2.743	3.563	7.04	2.146
v_{57}	2.267	2.281	2.575	3.263	3.157	3.463	3.078	3.075	3.002	3.011	4.441	5.936	1.042
v_{58}	1.868	2.598	2.892	2.132	2.026	5.438	3.395	3.514	3.351	3.981	3.31	3.328	2.765
v_{59}	5.005	3.905	3.712	3.627	3.521	6.223	3.34	6.373	6.3	2.55	4.805	8.674	4.34
v_{60}	7.201	5.447	4.827	4.897	5.554	10.747	5.651	8.832	8.693	6.71	4.64	7.423	8.197
v_{61}	2.739	0.804	1.404	1.319	1.213	4.02	1.907	4.169	4.096	2.825	2.497	6.408	2.136
v_{62}	2.796	1.162	1.456	1.371	1.265	4.348	1.38	4.214	4.705	1.215	2.549	6.465	2.181
v_{63}	2.523	0.254	0.548	1.106	0.727	5.528	1.051	4.288	4.125	2.218	1.753	6.177	2.466
v_{64}	2.705	0.318	0.682	1.534	0.909	5.079	1.185	4.336	4.197	2.074	1.887	6.096	3.34
v_{65}	7.08	9.466	9.831	9.382	9.283	10.033	10.334	6.397	5.821	10.31	10.56	11.237	7.367
v_{66}	1.671	1.181	1.475	1.286	1.18	5.215	1.978	3.317	3.154	2.564	2.464	4.81	2.473
v_{67}	4.116	6.502	6.796	6.418	6.312	7.55	7.299	3.861	3.285	7.773	7.596	8.273	4.83
v_{68}	3.786	6.172	6.466	6.088	5.982	8.273	6.969	4.583	4.007	7.555	7.266	7.943	5.553
v_{69}	4.601	2.711	3.626	3.223	3.117	5.819	2.474	5.969	5.896	1.684	4.371	8.27	3.936
v_{70}	3.133	1.456	1.146	1.423	1.317	6.703	1.97	4.779	4.43	2.541	0.861	7.019	3.935
v_{71}	2.968	1.581	1.141	1.079	0.973	6.538	1.965	4.614	4.451	2.536	0.838	7.014	3.865
v_{72}	3.391	1.285	0.921	1.429	1.323	6.961	1.745	5.037	4.874	2.316	0.349	6.794	4.215
v_{73}	5.591	4.944	5.238	6.986	6.88	2.985	6.037	6.552	5.976	5.247	8.164	9.68	4.434
v_{74}	6.275	7.754	8.048	7.67	7.564	3.669	8.551	7.236	6.66	6.945	8.848	10.364	5.118
v_{75}	4.29	1.903	2.49	2.754	2.648	5.358	1.528	5.224	5.151	2.114	3.235	7.475	3.191
v_{76}	3.288	1.493	2.08	2.344	2.238	4.84	1.118	4.706	4.633	0.695	2.825	6.957	2.673
v_{77}	4.221	1.834	2.421	2.685	2.579	6.335	1.459	5.867	5.704	2.082	3.166	7.756	4.045
v_{78}	2.646	1.012	1.306	1.221	1.115	4.275	1.809	4.064	3.991	2.395	2.399	6.315	2.031

v_{79}	5.42	2.577	4.127	4.042	3.936	6.638	3.756	6.788	6.715	2.966	5.22	9.089	4.755
v_{80}	4.095	1.708	2.295	2.559	2.453	6.209	1.333	5.741	5.578	1.956	3.04	7.63	3.919
v_{81}	4.552	2.272	1.908	2.817	2.711	8.122	2.346	6.198	6.035	3.303	1.797	7.664	4.483
v_{82}	0.508	1.987	2.281	1.903	1.797	3.97	2.784	2.045	1.882	3.37	3.081	4.597	1.25
v_{83}	5.707	5.368	5.662	7.102	6.996	3.101	6.461	6.668	6.092	5.671	8.28	9.796	4.55
v_{84}	4.567	3.286	3.274	3.189	3.083	5.785	2.721	5.935	5.862	1.931	4.618	8.236	3.902
v_{85}	4.016	2.429	2.723	2.638	2.532	5.234	2.352	5.384	5.311	1.562	3.816	7.685	3.351
v_{86}	3.397	1.72	1.41	1.687	1.581	6.967	2.234	5.043	4.88	2.805	1.125	7.283	4.199
v_{87}	3.13	1.742	1.776	2.37	1.746	4.673	2.182	4.142	4.515	1.805	2.981	6.76	2.442
v_{88}	5.078	4.486	4.849	7.063	6.373	2.448	5.352	6.068	5.492	5.723	7.845	9.218	3.845
v_{89}	4.111	3.257	3.62	6.096	5.406	1.481	4.123	5.101	4.525	4.494	6.878	8.251	2.878
v_{90}	3.371	1.31	0.996	2.2	1.575	5.518	0.679	5.002	4.863	1.511	2.201	6.762	2.649
v_{91}	3.011	1.294	1.657	2.251	1.627	5.299	2.16	4.07	4.382	2.477	2.862	6.641	2.43
v_{92}	2.216	0.692	1.055	0.682	0.399	5.762	1.558	3.847	3.708	2.447	1.465	5.587	2.851
v_{93}	1.234	2.16	2.523	2.502	1.66	4.78	3.026	2.865	2.726	3.915	2.964	4.313	1.853
v_{94}	4.852	2.791	2.477	3.681	3.056	6.436	1.97	6.483	6.344	1.082	3.682	8.243	4.245
v_{95}	5.115	3.054	2.74	4.435	3.319	7.45	2.091	6.746	6.607	2.497	4.178	10.044	5.75
v_{96}	4.532	2.702	2.082	2.228	2.335	8.078	1.488	6.163	6.024	3.107	1.971	7.837	5.172
v_{97}	0.547	2.935	3.298	3.44	2.75	5.001	3.801	3.086	2.947	4.69	4.222	5.595	2.305
v_{98}	5.825	4.971	5.334	7.81	7.12	3.195	5.837	6.815	6.239	6.208	8.592	9.965	4.592
v_{99}	3.873	1.812	1.192	1.772	1.879	7.419	1.56	5.504	5.365	2.752	1.515	7.411	4.508
v_{100}	4.565	2.313	1.693	1.768	1.875	7.618	2.06	5.703	5.564	3.252	1.511	7.377	4.712
v_{101}	5.399	3.012	3.599	3.863	3.757	7.218	2.447	7.045	6.882	1.765	4.344	8.934	5.335
v_{102}	4.563	2.176	2.763	3.027	2.921	6.382	1.611	6.209	6.046	0.896	3.508	8.098	4.499
v_{103}	4.841	2.454	3.041	3.305	3.199	6.66	1.889	6.487	6.324	1.207	3.786	8.376	4.777
v_{104}	5.693	3.306	3.893	4.157	4.051	7.512	2.741	7.339	7.176	2.059	4.638	9.228	5.629
v_{105}	2.116	1.325	1.619	0.84	0.734	5.686	2.122	3.762	3.599	2.708	2.018	4.602	3.012
v_{106}	2.232	0.486	0.78	0.611	0.505	5.802	1.283	3.878	3.715	1.869	1.789	5.767	3.034
v_{107}	2.344	1.146	1.677	0.506	0.4	5.965	1.788	4.041	3.878	2.374	1.223	5.314	3.292
v_{108}	2.695	1.497	1.326	0.857	0.751	6.316	2.192	4.392	4.229	2.725	0.872	5.665	3.643
v_{109}	2.436	1.619	1.586	1.95	0.492	6.057	1.88	4.133	3.97	2.466	1.132	5.071	3.384
v_{110}	4.754	3.146	3.509	5.663	4.973	2.124	4.012	5.744	5.168	4.383	4.714	7.912	2.192
v_{111}	2.437	4.825	5.188	5.33	4.64	6.891	5.691	5.933	5.357	6.58	6.112	6.595	4.195
v_{112}	5.118	3.087	3.114	4.318	3.693	4.808	2.607	5.654	6.024	1.789	4.319	8.748	3.951
v_{113}	4.179	2.426	1.806	1.875	1.982	7.725	2.629	5.81	5.671	3.688	1.618	5.487	4.819
v_{114}	8.32	6.566	5.946	6.016	6.123	11.866	6.77	9.951	9.812	7.829	5.759	8.542	8.96
v_{115}	0.775	3.163	3.526	3.668	2.978	5.229	4.029	3.314	3.175	4.918	4.45	4.859	2.533
v_{116}	5.519	3.458	3.144	4.348	3.723	5.765	2.637	6.611	6.981	1.819	4.349	8.91	4.908
v_{117}	4.437	2.376	2.062	3.266	2.641	5.189	1.555	5.266	5.639	0.737	3.267	7.828	3.566
v_{118}	0.764	3.269	3.632	3.774	3.084	4.147	4.135	1.12	0.981	5.024	4.556	5.929	1.451
v_{119}	2.148	3.075	3.438	3.179	2.102	5.694	3.941	3.779	3.64	4.83	3.46	3.356	2.768
v_{120}	5.989	5.839	7.183	7.252	5.104	8.459	6.705	6.544	6.405	7.594	6.995	2.634	5.532
v_{121}	9.228	10.154	11.497	11.566	9.419	12.774	11.02	10.859	10.72	11.909	11.309	4.231	9.847
v_{122}	2.48	3.96	4.323	4.465	3.775	4.703	4.826	1.067	0.491	5.903	5.247	6.62	2.007
v_{123}	1.662	1.559	1.922	3.043	2.353	4.354	2.425	3.293	3.229	2.846	3.825	5.292	1.277
v_{124}	3.22	1.503	1.498	2.702	2.077	5.508	1.181	4.279	4.591	0.922	2.703	6.85	2.639
v_{125}	2.575	3.501	3.864	3.843	2.784	6.121	4.367	4.206	4.067	5.256	4.142	4.582	3.194
v_{126}	2.789	3.585	3.948	4.774	4.084	1.689	4.451	3.779	3.203	5.452	5.556	6.929	1.556
v_{127}	3.006	1.289	1.652	2.246	1.622	4.614	2.058	4.129	4.502	1.792	2.857	6.636	2.429
v_{128}	4.172	2.56	2.246	3.45	2.825	4.632	1.739	4.708	5.081	0.921	3.451	7.802	3.008
v_{129}	3.617	1.787	1.167	1.313	1.42	7.163	1.747	5.248	5.109	2.939	1.056	6.922	4.257
NODES	v_{39}	v_{40}	v_{41}	v_{42}	v_{43}	v_{44}	v_{45}	v_{46}	v_{47}	v_{48}	v_{49}	v_{50}	v_{51}
v_0	6.188	4.004	5.774	15.446	8.148	4.629	5.47	5.741	8.588	7.652	5.148	5.032	11.991
v_1	2.406	1.626	2.202	11.181	3.883	0.837	0.884	1.252	4.122	2.89	2.56	2.115	7.121
v_2	1.982	2.893	3.442	11.696	4.711	2.101	0.978	1.281	3.941	3.479	2.406	2.78	6.48
v_3	2.872	2.321	4.168	12.586	5.601	2.803	1.868	2.171	5.017	4.369	1.626	1.982	7.37
v_4	3.571	4.18	5.133	13.285	6.3	3.69	2.567	2.87	4.409	5.068	3.485	3.841	6.682
v_5	2.694	0.548	3.156	12.135	4.837	1.791	1.976	2.561	5.431	4.341	1.058	0.943	7.996
v_6	6.214	8.937	6.823	3.43	5.049	7.542	7.355	6.986	8.88	4.999	9.853	9.408	10.217

v_7	8.451	4.742	6.678	16.3	9.335	4.921	6.039	6.623	11.832	8.907	6.996	5.6	13.169
v_8	6.9	9.653	7.611	3.679	5.821	8.279	7.776	7.774	9.807	5.798	10.163	10.196	11.005
v_9	2.736	3.325	4.319	12.223	4.795	2.803	1.753	2.056	3.594	4.254	2.844	2.946	5.867
v_{10}	3.04	3.526	4.622	12.527	5.098	3.106	2.056	2.359	3.898	4.281	3.045	3.147	6.171
v_{11}	2.805	4.447	5.31	12.292	5.845	3.667	2.617	2.84	3.663	4.046	3.966	4.068	7.614
v_{12}	3.513	1.569	2.342	11.694	4.382	1.197	1.823	2.191	5.061	3.901	2.485	2.04	8.216
v_{13}	3.548	4.338	1.408	10.721	4.121	2.542	3.737	3.85	6.712	3.625	4.848	4.733	7.964
v_{14}	3.112	4.496	1.351	8.751	3.997	2.503	3.577	3.852	6.494	3.516	5.412	4.967	7.831
v_{15}	0.485	3.139	2.097	9.256	2.227	2.49	1.659	1.29	3.904	1.019	4.055	3.61	5.241
v_{16}	2.287	3.456	0.688	9.021	3.171	1.84	2.752	2.967	5.668	2.897	4.587	4.142	7.005
v_{17}	3.702	4.311	13.169	13.4	6.431	3.821	2.698	3.001	5.847	5.199	3.616	3.972	8.2
v_{18}	3.332	3.96	4.812	13.066	6.081	3.471	2.348	2.651	5.497	4.849	3.265	3.621	7.85
v_{19}	4.273	3.484	5.331	13.739	7.002	3.966	3.268	3.572	6.418	5.77	2.789	3.066	8.771
v_{20}	1.92	2.949	1.003	9.611	2.314	1.722	2.109	2.317	4.905	1.818	4.033	3.344	6.301
v_{21}	2.555	3.322	4.015	11.997	5.284	2.601	1.551	1.854	4.7	4.052	2.627	3.001	7.053
v_{22}	3.621	4.283	5.203	13.355	6.37	3.687	2.637	2.94	4.479	4.862	3.802	3.904	6.752
v_{23}	1.413	4.255	3.433	11.455	4.47	4.152	1.8	1.431	2.264	3.262	5.169	5.076	4.178
v_{24}	3.635	2.175	4.081	12.317	5.578	2.393	2.648	2.655	5.177	4.853	1.539	1.796	7.854
v_{25}	4.426	5.872	3.83	6.422	4.906	4.498	5.032	5.144	7.497	4.41	6.382	6.267	8.725
v_{26}	1.989	3.507	1.393	8.859	2.454	2.112	2.454	2.707	4.951	1.973	4.423	3.978	6.288
v_{27}	3.064	1.12	1.893	11.245	3.933	0.748	1.374	1.742	4.612	3.452	2.036	1.591	7.767
v_{28}	3.358	0.913	2.187	11.539	4.227	1.042	1.668	2.036	4.906	3.746	2.079	1.473	8.061
v_{29}	2.985	1.971	1.408	11.161	3.849	0.263	1.583	1.951	6.346	3.368	2.887	2.442	7.683
v_{30}	2.879	1.865	1.302	11.055	3.743	0.157	1.477	1.845	6.24	3.262	2.781	2.336	7.577
v_{31}	3.772	6.393	4.963	11.866	4.837	5.682	3.735	3.366	2.345	3.62	7.309	6.864	1.977
v_{32}	3.861	0.985	2.69	12.042	4.73	1.545	2.585	2.888	5.705	4.249	1.329	0.723	8.564
v_{33}	2.743	5.153	3.039	8.176	1.241	3.758	3.884	3.515	5.912	1.394	6.069	5.624	7.249
v_{34}	2.67	4.99	2.876	7.6	1.281	3.595	3.811	3.442	5.336	1.231	5.906	5.461	6.673
v_{35}	3.407	1.913	3.276	12.089	5.316	2.131	2.42	2.723	4.915	3.843	1.432	1.534	7.188
v_{36}	4.163	1.553	2.586	12.339	5.027	1.441	2.761	3.129	7.524	4.546	2.824	2.218	8.861
v_{37}	5.658	7.42	3.886	13.016	6.543	5.031	6.123	6.376	9.04	6.062	9.674	8.278	10.377
v_{38}	0.637	3.331	2.289	9.146	2.117	3.009	1.851	1.482	3.794	0.9	4.247	3.802	5.131
v_{39}	0	2.925	2.381	9.541	2.512	3.101	1.07	0.701	2.819	1.295	3.841	3.396	5.092
v_{40}	2.925	0	2.942	12.294	4.982	1.797	2.423	2.791	5.661	4.501	1.779	1.173	8.816
v_{41}	2.381	2.942	0	10.2	2.887	1.145	2.468	2.721	5.384	2.406	4.303	3.858	6.721
v_{42}	9.541	12.294	10.2	0	8.479	10.971	10.784	10.415	12.309	8.429	13.282	12.837	13.646
v_{43}	2.512	4.982	2.887	8.479	0	3.673	3.799	3.43	5.324	1.309	5.984	5.539	6.661
v_{44}	3.101	1.797	1.145	10.971	3.673	0	1.634	2.002	6.104	3.126	2.938	2.493	7.441
v_{45}	1.07	2.423	2.468	10.784	3.799	1.634	0	0.369	3.239	2.171	3.053	2.948	5.512
v_{46}	0.701	2.791	2.721	10.415	3.43	2.002	0.369	0	2.87	1.925	3.421	2.949	5.143
v_{47}	2.819	5.661	5.384	12.309	5.324	6.104	3.239	2.87	0	4.202	6.268	5.796	4.264
v_{48}	1.295	4.501	2.406	8.429	1.309	3.126	2.171	1.925	4.202	0	5.488	5.043	5.429
v_{49}	3.841	1.779	4.303	13.282	5.984	2.938	3.053	3.421	6.268	5.488	0	0.518	8.268
v_{50}	3.396	1.173	3.858	12.837	5.539	2.493	2.948	2.949	5.796	5.043	0.518	0	9.211
v_{51}	5.092	8.816	6.721	13.646	6.661	7.441	5.512	5.143	4.264	5.429	8.268	9.211	0
v_{52}	2.97	2.021	3.868	12.847	5.549	2.503	1.901	2.269	5.116	4.184	1.326	1.682	7.469
v_{53}	1.671	2.165	2.209	10.512	3.527	1.672	0.642	1.011	3.858	2.295	2.675	2.56	6.475
v_{54}	1.941	4.411	2.316	7.907	0.584	3.036	2.475	2.571	4.848	0.724	4.921	4.806	6.1
v_{55}	2.879	3.79	4.339	12.593	5.608	2.998	1.875	2.178	5.024	4.376	3.773	3.68	7.377
v_{56}	1.365	3.742	3.385	11.079	4.094	2.436	0.683	0.766	3.51	2.862	3.725	3.632	5.863
v_{57}	0.431	3.33	2.281	9.975	2.99	3	1.185	0.816	2.944	1.758	3.84	3.725	5.297
v_{58}	2.857	3.647	0.717	10.03	3.429	1.869	3.046	3.159	6.02	2.933	4.157	4.042	7.272
v_{59}	3.559	4.325	5.019	13.273	6.288	3.678	2.555	2.858	5.704	5.056	3.63	3.986	8.057
v_{60}	7.883	4.812	6.306	16.059	8.747	5.437	6.495	6.863	11.244	8.46	7.066	5.84	12.799
v_{61}	1.355	2.159	2.753	11.069	4.084	1.37	0.285	0.654	3.501	2.852	2.669	2.554	5.854
v_{62}	1.683	2.214	2.81	11.114	4.129	1.422	0.614	0.982	3.829	2.897	2.197	2.104	6.182
v_{63}	1.303	2.522	2.522	11.501	4.203	1.157	1.342	1.613	4.46	3.588	2.498	1.698	7.362
v_{64}	3.382	1.438	2.211	10.95	4.251	1.066	0.893	2.06	4.93	2.733	2.354	1.909	8.085
v_{65}	7.762	10.515	8.421	3.967	6.683	9.141	8.638	8.392	10.669	6.66	11.025	10.91	11.921
v_{66}	2.565	2.23	1.155	10.53	3.232	1.206	1.675	2.867	5.823	2.736	2.74	2.625	7.075

v_{67}	5.225	7.551	5.457	5.319	4.147	6.177	6.101	5.855	8.132	4.124	8.061	7.946	9.384
v_{68}	5.948	7.221	5.127	5.073	4.869	5.847	6.824	6.578	8.855	4.846	7.731	7.616	10.107
v_{69}	3.155	3.459	4.615	12.869	5.884	3.274	2.151	2.454	5.3	4.652	2.764	3.12	7.653
v_{70}	4.027	1.494	2.449	11.992	4.694	1.304	2.285	2.556	7.285	4.198	2.275	2.159	8.537
v_{71}	3.957	1.489	2.001	11.827	4.529	0.856	2.41	2.681	7.12	4.033	2.27	2.154	8.372
v_{72}	4.307	1.269	2.351	12.25	4.952	1.206	2.372	2.643	7.543	4.456	2.05	1.934	8.795
v_{73}	3.151	5.993	6.024	12.949	5.964	6.744	3.571	3.202	0.712	4.732	6.327	6.683	4.819
v_{74}	4.849	8.803	6.708	13.633	6.648	7.428	5.269	4.9	3.135	5.416	9.131	8.381	5.503
v_{75}	2.693	2.362	3.82	12.124	5.139	2.805	1.624	1.992	4.839	3.907	2.444	2.252	7.192
v_{76}	2.175	1.952	3.302	11.606	4.621	2.395	1.106	1.474	4.321	3.389	1.677	2.033	6.674
v_{77}	3.67	2.293	4.101	13.08	3.65	2.736	2.601	2.969	5.816	5.286	2.375	2.183	8.169
v_{78}	1.356	2.061	2.66	10.964	3.979	1.272	0.638	0.909	3.756	2.747	2.571	2.456	6.109
v_{79}	3.974	4.741	5.434	13.688	6.703	4.093	2.97	3.273	6.119	5.471	4.046	4.402	8.472
v_{80}	3.544	2.167	3.975	12.954	5.656	2.61	2.475	2.843	5.69	5.16	2.249	2.057	8.043
v_{81}	4.077	1.362	3.739	13.411	6.113	2.594	3.359	3.63	6.477	5.617	2.526	2.1	9.956
v_{82}	1.645	3.036	0.941	9.08	1.96	1.661	2.195	2.275	4.552	1.464	3.546	3.431	5.804
v_{83}	3.575	6.417	6.14	13.065	6.08	6.86	3.995	3.626	0.822	4.848	7.389	7.107	4.935
v_{84}	3.121	3.706	4.581	12.835	5.85	3.24	2.117	2.42	5.266	4.618	3.011	3.367	7.619
v_{85}	2.57	3.337	4.03	12.284	5.299	2.689	1.566	1.869	4.715	4.067	2.642	2.998	7.068
v_{86}	4.291	1.758	2.713	12.256	4.958	1.568	2.549	2.82	7.549	4.462	2.539	2.423	8.801
v_{87}	2.002	3.017	3.105	11.489	4.057	1.903	1.015	1.318	4.188	3.243	3.011	2.684	6.461
v_{88}	3.092	5.605	5.563	12.466	5.437	6.282	3.496	3.127	0.735	4.22	6.548	6.076	4.282
v_{89}	1.863	4.376	4.596	11.499	4.47	5.315	2.267	1.898	1.043	3.253	5.166	4.847	3.315
v_{90}	2.893	1.514	2.877	11.575	4.917	1.732	1.906	2.209	5.079	3.329	1.849	1.403	7.352
v_{91}	2.674	2.413	2.986	11.356	3.985	1.784	1.687	1.99	4.86	3.11	3.329	2.884	7.133
v_{92}	2.893	1.975	1.701	11.074	3.762	0.556	1.423	1.791	6.259	3.281	2.727	2.282	7.596
v_{93}	1.895	3.279	0.658	10.093	2.78	1.449	2.36	2.613	5.277	2.299	4.195	3.75	6.614
v_{94}	4.489	2.995	4.358	13.171	6.398	3.213	3.502	3.805	5.997	4.925	2.514	2.616	8.27
v_{95}	5.792	2.542	4.621	13.973	6.661	3.476	4.418	4.721	7.011	6.18	1.226	1.77	9.284
v_{96}	5.214	1.536	3.637	13.39	6.078	2.492	3.75	4.118	7.621	5.597	1.836	0.857	9.912
v_{97}	2.536	4.054	1.94	8.568	3.001	2.659	3.001	3.254	5.498	2.52	4.97	4.525	6.835
v_{98}	3.577	6.09	6.31	13.213	6.184	7.029	3.981	3.612	0.92	4.967	7.033	6.561	5.029
v_{99}	4.55	0.576	3.181	12.731	5.419	2.036	2.86	3.228	6.098	4.938	2.738	1.749	9.253
v_{100}	4.754	1.076	3.177	12.93	5.618	2.032	3.361	3.729	6.599	5.137	2.792	1.815	9.452
v_{101}	4.554	3.432	5.279	14.268	6.96	3.914	3.846	4.149	6.375	5.269	2.892	2.994	8.648
v_{102}	3.718	2.596	4.443	13.432	6.124	3.078	2.714	3.017	5.863	5.215	1.901	2.257	7.812
v_{103}	3.996	2.874	4.721	13.71	6.402	3.356	2.992	3.295	6.141	5.493	2.179	2.535	8.494
v_{104}	4.848	3.726	5.573	14.562	7.254	4.208	3.844	4.147	6.993	6.345	3.031	3.387	9.346
v_{105}	3.104	2.374	0.716	10.975	3.677	0.577	3.293	3.406	6.268	3.181	2.884	2.769	7.52
v_{106}	3.126	1.535	2.112	11.091	3.793	0.662	1.155	1.426	6.384	3.297	2.045	1.93	7.636
v_{107}	3.384	2.067	1.428	11.254	3.956	0.283	1.821	2.092	6.547	3.46	2.848	2.732	7.799
v_{108}	3.735	1.716	1.779	11.605	4.307	0.634	2.172	2.443	6.898	3.811	2.497	2.381	8.15
v_{109}	3.476	2.513	1.52	11.367	4.048	0.375	1.913	3.799	6.66	3.552	3.023	2.908	7.912
v_{110}	1.752	4.265	4.257	12.142	5.113	4.977	2.156	1.787	1.685	3.896	5.208	4.736	3.958
v_{111}	4.426	5.944	3.83	6.422	4.891	4.549	4.891	5.144	7.388	4.41	6.86	6.415	8.725
v_{112}	3.511	3.632	5.093	12.998	5.569	3.85	2.527	2.83	4.369	4.752	3.151	3.253	6.642
v_{113}	4.861	1.959	3.284	13.037	5.725	2.139	3.473	3.841	8.222	5.244	4.213	2.818	9.559
v_{114}	9.002	5.931	7.425	17.178	9.866	6.28	7.614	7.982	12.363	9.385	8.185	6.959	13.7
v_{115}	2.764	4.282	1.204	8.872	3.229	2.357	3.229	3.482	5.726	2.748	5.198	4.753	7.063
v_{116}	4.468	3.662	5.025	13.955	6.526	3.88	3.484	3.787	5.326	5.709	3.181	3.283	7.599
v_{117}	3.126	2.58	3.943	12.613	5.181	2.798	2.139	2.442	4.75	4.367	2.099	2.201	7.023
v_{118}	1.978	4.388	2.274	8.179	1.035	2.993	3.119	2.75	4.644	0.63	5.304	4.859	5.981
v_{119}	2.81	4.295	0.8	11.007	3.694	1.945	3.275	3.528	6.191	3.213	5.11	4.665	7.528
v_{120}	5.574	7.337	3.802	10.391	6.459	4.947	6.039	6.292	8.956	5.978	9.591	8.195	10.293
v_{121}	9.889	11.651	8.117	17.247	10.774	9.262	10.354	10.607	13.271	10.293	13.905	12.509	14.608
v_{122}	2.534	5.079	2.965	7.284	1.37	3.684	3.675	3.306	5.2	1.32	5.995	5.55	6.537
v_{123}	1.163	2.678	1.637	10.203	3.208	2.357	0.556	0.669	3.683	1.957	3.594	3.149	6.188
v_{124}	2.883	2.016	3.195	11.565	4.194	2.234	1.896	2.199	5.069	3.319	1.919	2.021	7.342
v_{125}	3.236	4.62	1.475	9.107	4.121	2.627	3.701	3.954	6.618	3.64	5.536	5.091	7.955
v_{126}	2.083	4.704	3.274	10.177	3.148	3.993	3.224	2.855	2.186	1.931	5.62	5.175	3.523

v_{127}	1.989	2.893	2.981	11.476	4.044	1.779	1.002	1.305	4.175	3.23	2.887	2.782	6.448
v_{128}	2.568	2.764	4.147	12.055	4.623	2.982	1.581	1.884	4.193	3.809	2.283	2.385	6.466
v_{129}	4.299	0.763	2.722	12.475	5.163	1.577	2.835	3.203	6.073	4.682	2.751	1.774	8.997
NODES	v_{52}	v_{53}	v_{54}	v_{55}	v_{56}	v_{57}	v_{58}	v_{59}	v_{60}	v_{61}	v_{62}	v_{63}	v_{64}
v_0	5.556	5.128	7.577	6.817	6.273	5.757	6.736	7.884	1.424	5.423	5.475	4.567	5.341
v_1	2.125	0.626	3.262	2.248	1.686	1.791	2.738	2.928	5.698	0.62	0.672	0.779	0.508
v_2	1.492	1.394	4.14	1.21	1.301	1.548	3.917	1.577	6.962	1.383	0.679	1.792	1.343
v_3	0.368	2.302	5.03	2.101	2.191	2.438	4.643	2.115	6.855	2.273	1.569	2.103	2.264
v_4	2.227	2.983	5.729	2.52	2.89	3.137	5.609	2.045	8.714	2.972	3.41	3.381	4.078
v_5	1.824	1.935	4.266	3.809	2.779	3.1	3.631	4.152	5.215	1.929	1.984	1.073	1.847
v_6	9.418	7.083	4.477	8.044	7.65	6.546	7.247	9.844	12.63	7.64	7.685	8.072	7.521
v_7	6.315	5.697	8.764	7.386	7.057	8.729	6.121	8.619	4.228	5.991	6.043	5.136	5.909
v_8	9.556	8.144	5.25	8.833	8.438	7.579	8.035	8.358	13.418	8.428	8.473	8.74	8.922
v_9	1.658	2.132	4.224	1.705	2.076	2.868	4.795	1.23	7.634	2.158	2.593	2.567	2.118
v_{10}	1.611	2.435	4.527	2.009	2.379	3.172	4.995	1.534	8.098	2.461	2.794	2.87	3.462
v_{11}	1.749	2.996	5.274	1.774	2.94	2.937	5.786	1.299	8.235	3.022	3.35	3.431	2.982
v_{12}	2.05	1.565	3.811	3.187	2.625	2.73	3.047	4.354	6.893	1.559	1.611	0.703	0.201
v_{13}	4.814	3.078	3.55	5.659	4.416	3.54	0.839	6.185	6.851	3.919	3.976	3.688	3.812
v_{14}	4.977	3.318	3.551	5.448	4.494	3.39	1.278	6.128	8.64	3.862	3.919	3.631	3.755
v_{15}	3.276	1.387	1.656	3.468	1.954	0.85	2.573	1.8	7.651	2.004	1.989	2.274	1.825
v_{16}	4.152	2.493	2.6	4.623	3.669	2.565	1.163	5.303	7.001	3.037	3.094	2.806	2.725
v_{17}	2.358	3.114	5.169	2.651	3.021	3.268	5.637	2.176	8.845	3.103	3.541	3.512	3.063
v_{18}	2.007	2.764	5.51	2.301	2.671	3.464	5.287	1.826	8.494	2.673	3.19	3.162	2.713
v_{19}	1.531	3.465	6.431	3.222	3.591	3.839	5.806	2.746	8.018	3.673	2.713	3.169	3.33
v_{20}	3.598	1.45	1.743	4.031	2.788	1.877	1.331	3.556	6.714	2.349	2.406	2.036	2.242
v_{21}	1.713	1.967	4.713	1.479	1.874	2.642	4.593	1.004	7.856	1.876	2.197	2.278	1.941
v_{22}	2.616	3.053	5.108	2.59	2.96	3.753	5.679	2.115	8.76	3.042	3.37	3.451	3.002
v_{23}	3.556	3.331	3.899	2.787	2.128	1.978	4.377	2.312	8.327	2.085	2.413	3.044	2.595
v_{24}	0.27	2.215	5.007	2.585	2.675	2.922	4.243	2.812	6.768	3.053	1.443	2.276	2.06
v_{25}	6.348	4.373	4.335	6.954	5.711	4.835	3.608	6.479	9.637	5.176	5.233	5.079	5.069
v_{26}	3.988	2.195	1.883	4.325	3.371	2.267	1.868	5.005	7.201	2.739	2.796	2.523	2.705
v_{27}	1.601	1.116	3.362	2.738	2.176	2.281	2.598	3.905	5.447	0.804	1.162	0.254	0.318
v_{28}	2.188	1.41	3.656	3.032	2.47	2.575	2.892	3.712	4.827	1.404	1.456	0.548	0.682
v_{29}	2.452	1.621	3.278	2.947	2.385	3.263	2.132	3.627	4.897	1.319	1.371	1.106	1.534
v_{30}	2.346	1.515	3.172	2.841	2.279	3.157	2.026	3.521	5.554	1.213	1.265	0.727	0.909
v_{31}	5.635	4.641	4.266	5.543	4.029	3.463	5.438	6.223	10.747	4.02	4.348	5.528	5.079
v_{32}	1.036	1.913	4.159	2.956	2.908	3.078	3.395	3.34	5.651	1.907	1.38	1.051	1.185
v_{33}	5.634	3.612	0.656	5.693	4.179	3.075	3.514	6.373	8.832	4.169	4.214	4.288	4.336
v_{34}	5.471	3.539	0.709	5.62	4.106	3.002	3.351	6.3	8.693	4.096	4.705	4.125	4.197
v_{35}	0.246	1.846	4.745	2.791	2.743	3.011	3.981	2.55	6.71	2.825	1.215	2.218	2.074
v_{36}	2.933	2.799	4.456	4.125	3.563	4.441	3.31	4.805	4.64	2.497	2.549	1.753	1.887
v_{37}	7.523	5.864	5.972	7.994	7.04	5.936	3.328	8.674	7.423	6.408	6.465	6.177	6.096
v_{38}	3.468	1.579	1.546	3.66	2.146	1.042	2.765	4.34	8.197	2.136	2.181	2.466	3.34
v_{39}	2.97	1.671	1.941	2.879	1.365	0.431	2.857	3.559	7.883	1.355	1.683	1.303	3.382
v_{40}	2.021	2.165	4.411	3.79	3.742	3.33	3.647	4.325	4.812	2.159	2.214	2.522	1.438
v_{41}	3.868	2.209	2.316	4.339	3.385	2.281	0.717	5.019	6.306	2.753	2.81	2.522	2.211
v_{42}	12.847	10.512	7.907	12.593	11.079	9.975	10.03	13.273	16.059	11.069	11.114	11.501	10.95
v_{43}	5.549	3.527	0.584	5.608	4.094	2.99	3.429	6.288	8.747	4.084	4.129	4.203	4.251
v_{44}	2.503	1.672	3.036	2.998	2.436	3	1.869	3.678	5.437	1.37	1.422	1.157	1.066
v_{45}	1.901	0.642	2.475	1.875	0.683	1.185	3.046	2.555	6.495	0.285	0.614	1.342	0.893
v_{46}	2.269	1.011	2.571	2.178	0.766	0.816	3.159	2.858	6.863	0.654	0.982	1.613	2.06
v_{47}	5.116	3.858	4.848	5.024	3.51	2.944	6.02	5.704	11.244	3.501	3.829	4.46	4.93
v_{48}	4.184	2.295	0.724	4.376	2.862	1.758	2.933	5.056	8.46	2.852	2.897	3.588	2.733
v_{49}	1.326	2.675	4.921	3.773	3.725	3.84	4.157	3.63	7.066	2.669	2.197	2.498	2.354
v_{50}	1.682	2.56	4.806	3.68	3.632	3.725	4.042	3.986	5.84	2.554	2.104	1.698	1.909
v_{51}	7.469	6.475	6.1	7.377	5.863	5.297	7.272	8.057	12.799	5.854	6.182	7.362	8.085
v_{52}	0	1.954	4.887	2.359	2.449	3.245	4.123	2.373	6.555	2.531	1.323	1.779	1.919
v_{53}	1.954	0	2.911	2.291	1.734	2.179	2.387	2.971	6.237	0.663	0.715	1	1.434
v_{54}	4.887	2.911	0	5.037	3.523	2.419	2.858	5.717	8.176	3.513	3.558	3.498	3.68

v_{55}	2.359	2.291	5.037	0	2.198	2.119	4.968	0.475	7.859	2.28	2.608	2.689	3.056
v_{56}	2.449	1.734	3.523	2.198	0	1.48	3.725	2.878	7.297	1.088	1.416	2.145	1.696
v_{57}	3.245	2.179	2.419	2.119	1.48	0	2.849	3.125	8.161	0.859	1.187	2.417	2.599
v_{58}	4.123	2.387	2.858	4.968	3.725	2.849	0	5.494	7.543	3.228	3.285	2.997	3.121
v_{59}	2.373	2.971	5.717	0.475	2.878	3.125	5.494	0	8.859	2.88	2.529	3.369	4.223
v_{60}	6.555	6.237	8.176	7.859	7.297	8.161	7.543	8.859	0	6.231	6.283	5.375	5.509
v_{61}	2.531	0.663	3.513	2.28	1.088	0.859	3.228	2.88	6.231	0	0.624	1.058	1.428
v_{62}	1.323	0.715	3.558	2.608	1.416	1.187	3.285	2.529	6.283	0.624	0	1.893	1.48
v_{63}	1.779	1	3.498	2.689	2.145	2.417	2.997	3.369	5.375	1.058	1.893	0	0.572
v_{64}	1.919	1.434	3.68	3.056	1.696	2.599	3.121	4.223	5.509	1.428	1.48	0.572	0
v_{65}	10.418	9.006	6.112	9.695	9.056	8.441	8.897	9.22	14.28	9.341	9.9	9.602	9.784
v_{66}	2.706	1.057	2.661	3.103	2.478	2.557	1.63	4.201	6.184	1.391	2.307	1.317	1.499
v_{67}	7.881	6.469	3.576	7.158	6.519	5.904	5.933	6.683	11.316	6.804	7.363	6.638	6.635
v_{68}	7.697	5.722	4.298	7.881	7.242	6.627	5.603	7.406	10.986	6.057	8.086	6.308	6.49
v_{69}	1.506	2.567	5.313	2.104	2.474	2.721	5.09	1.629	7.993	2.476	1.852	3.15	3.357
v_{70}	2.683	1.943	4.123	3.632	3.088	4.019	3.325	4.312	4.581	2.001	2.836	1.592	1.774
v_{71}	2.678	2.068	3.958	3.757	3.213	3.949	2.725	5.593	4.576	2.126	2.961	1.717	1.899
v_{72}	2.458	2.03	4.381	3.719	3.279	2.659	3.075	4.786	4.356	2.088	2.923	1.85	1.603
v_{73}	5.07	5.069	5.393	3.177	3.866	3.716	6.5	2.702	11.884	4.406	4.71	5.08	3.886
v_{74}	6.768	6.757	6.077	4.875	5.564	5.414	7.184	4.4	12.568	6.104	6.408	7.89	8.072
v_{75}	2.256	1.725	4.568	2.605	2.557	2.197	4.295	3.285	6.857	1.634	1.809	2.365	2.221
v_{76}	0.803	1.207	4.05	2.087	2.039	1.679	3.777	2.767	6.447	1.116	1.291	1.955	1.811
v_{77}	2.19	2.579	5.211	3.582	3.534	3.174	4.576	4.262	6.788	2.611	2.786	2.296	2.152
v_{78}	2.537	0.611	3.408	2.633	1.441	0.925	3.135	2.456	6.133	0.354	1.963	1.148	1.33
v_{79}	2.789	3.386	6.132	0.896	3.293	3.54	5.909	0.421	9.275	3.295	2.671	3.969	4.639
v_{80}	2.064	2.453	5.085	3.456	3.408	3.048	4.45	4.136	6.662	2.485	2.66	2.17	2.026
v_{81}	3.445	3.017	5.542	4.706	4.162	3.646	4.463	5.773	4.568	3.075	3.91	2.734	2.59
v_{82}	3.512	1.536	1.389	3.578	2.939	1.998	1.417	3.103	6.801	1.871	3.396	2.123	2.305
v_{83}	5.494	5.493	5.509	3.601	4.29	4.14	6.616	3.126	12	4.83	5.134	5.504	5.686
v_{84}	1.754	2.533	5.279	2.07	2.44	2.687	5.056	1.595	8.24	2.442	1.818	3.116	3.604
v_{85}	1.384	1.982	4.728	1.831	1.889	2.136	4.505	1.356	7.55	1.891	1.267	2.565	2.747
v_{86}	2.947	2.207	4.387	3.896	3.352	4.283	3.589	4.576	4.845	2.265	3.1	1.856	2.038
v_{87}	1.859	1.231	3.486	1.386	1.338	2.134	3.581	2.066	6.603	1.42	1.748	1.667	1.218
v_{88}	5.396	5.241	4.866	5.304	3.79	3.224	6.038	5.984	11.565	3.781	4.109	4.74	4.291
v_{89}	4.167	2.909	3.899	4.075	2.561	1.995	5.071	4.755	8.447	2.552	2.88	3.511	3.062
v_{90}	1.356	1.332	4.346	2.277	2.229	2.497	3.582	2.957	5.823	2.311	0.701	1.238	1.318
v_{91}	2.568	1.113	3.414	2.058	2.01	2.278	3.462	2.738	6.484	2.092	0.482	1.548	1.099
v_{92}	2.292	1.165	3.191	2.787	2.225	3.171	2.407	3.467	5.185	1.159	1.211	0.946	0.994
v_{93}	3.76	2.101	2.209	4.231	3.277	2.173	1.133	4.911	6.886	2.645	2.702	2.414	2.538
v_{94}	1.328	2.928	5.827	3.873	3.825	4.093	5.063	3.632	7.304	3.907	2.297	2.719	2.914
v_{95}	2.342	3.844	6.09	4.789	4.741	5.009	5.326	4.646	7.437	3.838	3.213	2.982	3.83
v_{96}	2.952	3.492	5.507	4.445	4.397	4.657	4.361	5.256	5.23	3.486	2.869	2.63	3.404
v_{97}	4.535	2.742	2.43	4.872	3.918	2.814	2.415	5.552	7.942	3.286	3.343	3.189	3.179
v_{98}	5.881	4.623	5.613	5.789	4.275	3.709	6.785	6.469	10.161	4.266	4.594	5.225	4.776
v_{99}	2.597	2.602	4.848	4.366	3.662	3.767	3.905	4.901	4.804	2.596	2.79	1.74	2.514
v_{100}	3.097	3.103	5.047	4.866	4.163	4.268	3.901	5.401	4.77	3.097	3.29	2.241	3.015
v_{101}	1.706	3.272	6.205	4.217	4.169	4.437	5.441	4.01	7.682	4.251	2.641	3.097	3.258
v_{102}	0.87	2.436	5.369	3.381	3.333	3.601	4.605	3.174	6.846	3.415	1.805	2.261	2.422
v_{103}	0.91	2.855	5.831	3.225	3.611	3.879	4.883	3.452	7.124	3.693	2.083	2.539	2.7
v_{104}	1.762	3.707	6.683	4.077	4.465	4.733	5.737	4.306	7.978	4.547	2.937	3.393	3.554
v_{105}	2.85	2.634	3.106	3.502	4.108	3.004	1.44	4.255	6.014	1.947	1.999	1.734	1.781
v_{106}	2.011	0.813	3.222	2.502	1.73	2.482	2.69	2.972	5.641	0.664	0.716	0.674	0.552
v_{107}	2.516	1.479	3.385	3.168	2.679	3.283	2.152	3.921	5.154	1.613	1.665	1.4	1.447
v_{108}	2.867	1.83	3.736	3.519	3.03	3.634	2.503	4.272	4.803	1.964	2.016	1.751	1.798
v_{109}	2.989	3.027	3.477	3.641	2.771	3.375	2.244	4.013	5.063	1.705	1.757	1.492	1.539
v_{110}	4.056	2.798	4.542	3.964	2.45	1.884	4.733	4.644	8.336	2.441	2.769	3.4	2.951
v_{111}	6.425	4.632	4.32	6.762	5.808	4.704	3.608	7.442	9.832	5.176	5.233	5.079	5.069
v_{112}	1.965	2.906	4.998	2.48	2.85	3.643	5.569	2.005	7.941	2.932	2.9	3.356	2.892
v_{113}	3.533	3.215	5.154	4.837	4.275	5.139	5.607	5.837	4.689	3.209	3.261	2.354	3.127
v_{114}	7.674	7.356	9.295	8.978	8.416	9.28	8.662	9.978	1.119	7.35	7.402	6.494	7.268

v_{115}	4.763	2.97	2.658	5.1	4.146	3.042	1.679	5.78	8.17	3.514	3.571	3.417	3.407
v_{116}	1.995	3.561	5.955	3.437	3.807	4.6	5.73	2.962	7.971	3.889	2.93	3.386	3.547
v_{117}	0.913	2.479	4.61	2.372	2.462	3.258	4.648	2.385	6.889	2.544	1.848	2.304	2.465
v_{118}	4.869	2.847	0.464	4.928	3.414	2.31	2.749	5.608	8.276	3.404	3.449	3.523	3.285
v_{119}	4.675	3.016	3.123	5.146	4.192	3.088	0.71	5.826	7.571	3.56	3.617	3.329	3.453
v_{120}	7.439	5.78	5.888	7.91	6.956	5.852	3.244	8.59	7.341	6.324	6.381	6.093	6.217
v_{121}	11.754	10.095	10.203	12.225	11.271	10.167	7.559	12.905	11.655	10.639	10.696	10.408	10.532
v_{122}	5.292	3.403	0.798	5.484	3.97	2.866	3.44	6.164	8.967	3.96	4.005	4.214	3.841
v_{123}	2.456	0.927	2.637	2.427	1.235	1.125	2.113	3.107	7.545	0.841	1.169	1.813	1.364
v_{124}	0.767	1.322	3.623	2.267	2.219	2.487	3.671	2.947	6.325	2.301	0.691	1.74	1.308
v_{125}	5.101	3.442	3.55	5.572	4.618	3.514	1.402	6.252	8.797	3.986	4.043	3.755	3.879
v_{126}	4.841	2.952	2.577	5.033	3.519	2.415	3.749	5.713	9.276	3.509	3.554	3.839	3.39
v_{127}	1.735	1.107	3.473	1.373	1.325	2.121	3.457	2.053	6.479	1.407	1.735	1.543	1.094
v_{128}	1.097	1.96	4.052	1.814	1.904	2.7	4.623	1.828	7.073	1.986	2.032	2.488	1.946
v_{129}	2.784	2.577	4.592	4.199	3.637	3.742	3.446	5.088	4.315	2.571	2.623	1.715	2.489
NODES	v_{65}	v_{66}	v_{67}	v_{68}	v_{69}	v_{70}	v_{71}	v_{72}	v_{73}	v_{74}	v_{75}	v_{76}	v_{77}
v_0	13.472	5.376	10.508	10.178	7.185	3.773	3.768	3.548	11.076	11.76	6.049	5.639	5.98
v_1	9.357	1.022	6.82	6.073	2.524	1.551	1.676	1.809	4.454	7.108	1.682	1.164	2.358
v_2	9.673	2.206	7.136	7.859	1.174	2.735	2.86	2.944	4.273	5.971	1.708	1.19	2.685
v_3	10.563	3.03	8.026	8.021	1.318	3.007	3.002	2.782	4.812	6.51	2.58	1.127	2.514
v_4	11.259	3.795	8.722	9.445	0.727	4.77	4.765	4.545	4.741	6.439	3.297	2.89	4.277
v_5	10.285	2	7.321	6.991	3.268	1.534	1.529	1.309	5.763	8.573	2.132	1.722	2.063
v_6	1.651	7.101	1.889	1.644	9.44	8.563	8.398	8.821	9.52	10.204	8.695	8.177	9.651
v_7	14.521	5.944	11.557	11.227	7.753	4.341	4.336	4.116	12.472	13.156	6.617	6.207	6.548
v_8	1.9	7.889	2.678	2.432	10.228	9.351	9.186	9.609	10.308	10.992	9.483	8.965	10.439
v_9	10.444	2.981	7.907	8.63	0.473	3.51	3.948	3.728	3.926	5.624	2.483	2.073	3.46
v_{10}	10.748	3.284	8.211	8.934	0.095	4.154	4.149	3.929	4.23	5.928	2.786	2.274	3.661
v_{11}	10.513	3.845	7.976	8.699	1.482	4.374	5.07	4.85	3.995	5.693	3.347	2.829	4.324
v_{12}	9.915	1.63	6.951	6.621	3.488	1.905	2.03	1.734	5.393	8.203	2.352	1.942	2.283
v_{13}	9.588	2.321	6.624	6.294	5.781	3.846	3.398	3.748	7.191	7.875	4.986	4.468	5.267
v_{14}	6.972	2.264	4.008	3.678	5.724	3.959	3.359	3.709	7.134	7.818	4.929	4.411	5.21
v_{15}	7.477	2.281	4.94	5.663	3.744	3.743	3.673	4.023	4.544	5.228	2.999	2.481	3.853
v_{16}	7.242	1.439	4.278	3.948	4.899	3.134	2.697	3.047	6.308	6.992	4.104	3.586	4.385
v_{17}	11.39	3.926	8.853	9.576	0.858	4.901	4.896	4.676	4.872	6.57	3.428	3.021	4.408
v_{18}	11.04	3.576	8.503	9.226	0.508	4.55	4.545	4.325	4.522	6.22	3.078	2.67	4.057
v_{19}	11.96	4.096	9.423	9.087	1.196	4.073	4.068	3.848	5.442	7.14	3.646	2.193	3.58
v_{20}	8.001	1.281	5.037	4.707	4.211	2.743	2.578	3.001	5.604	6.288	3.416	2.898	3.831
v_{21}	10.194	2.704	7.756	8.355	1.179	3.251	3.946	3.854	3.629	5.448	2.231	1.874	3.369
v_{22}	11.576	3.865	8.792	9.515	0.796	4.394	4.906	4.686	4.811	6.509	3.367	2.849	4.344
v_{23}	9.729	4.085	7.192	7.915	2.468	5.547	5.477	4.074	2.586	4.165	3.423	2.905	4.4
v_{24}	10.538	2.826	8.001	7.817	1.946	2.803	2.798	2.578	5.509	7.207	2.376	0.923	2.31
v_{25}	4.643	4.108	1.679	1.349	7.038	5.57	5.405	5.828	8.028	8.712	6.727	5.725	6.658
v_{26}	7.08	1.671	4.116	3.786	4.601	3.133	2.968	3.391	5.591	6.275	4.29	3.288	4.221
v_{27}	9.466	1.181	6.502	6.172	2.711	1.456	1.581	1.285	4.944	7.754	1.903	1.493	1.834
v_{28}	9.831	1.475	6.796	6.466	3.626	1.146	1.141	0.921	5.238	8.048	2.49	2.08	2.421
v_{29}	9.382	1.286	6.418	6.088	3.223	1.423	1.079	1.429	6.986	7.67	2.754	2.344	2.685
v_{30}	9.283	1.18	6.312	5.982	3.117	1.317	0.973	1.323	6.88	7.564	2.648	2.238	2.579
v_{31}	10.033	5.215	7.55	8.273	5.819	6.703	6.538	6.961	2.985	3.669	5.358	4.84	6.335
v_{32}	10.334	1.978	7.299	6.969	2.474	1.97	1.965	1.745	6.037	8.551	1.528	1.118	1.459
v_{33}	6.397	3.317	3.861	4.583	5.969	4.779	4.614	5.037	6.552	7.236	5.224	4.706	5.867
v_{34}	5.821	3.154	3.285	4.007	5.896	4.43	4.451	4.874	5.976	6.66	5.151	4.633	5.704
v_{35}	10.31	2.564	7.773	7.555	1.684	2.541	2.536	2.316	5.247	6.945	2.114	0.695	2.082
v_{36}	10.56	2.464	7.596	7.266	4.371	0.861	0.838	0.349	8.164	8.848	3.235	2.825	3.166
v_{37}	11.237	4.81	8.273	7.943	8.27	7.019	7.014	6.794	9.68	10.364	7.475	6.957	7.756
v_{38}	7.367	2.473	4.83	5.553	3.936	3.935	3.865	4.215	4.434	5.118	3.191	2.673	4.045
v_{39}	7.762	2.565	5.225	5.948	3.155	4.027	3.957	4.307	3.151	4.849	2.693	2.175	3.67
v_{40}	10.515	2.23	7.551	7.221	3.459	1.494	1.489	1.269	5.993	8.803	2.362	1.952	2.293
v_{41}	8.421	1.155	5.457	5.127	4.615	2.449	2.001	2.351	6.024	6.708	3.82	3.302	4.101
v_{42}	3.967	10.53	5.319	5.073	12.869	11.992	11.827	12.25	12.949	13.633	12.124	11.606	13.08

v_{43}	6.683	3.232	4.147	4.869	5.884	4.694	4.529	4.952	5.964	6.648	5.139	4.621	3.65
v_{44}	9.141	1.206	6.177	5.847	3.274	1.304	0.856	1.206	6.744	7.428	2.805	2.395	2.736
v_{45}	8.638	1.675	6.101	6.824	2.151	2.285	2.41	2.372	3.571	5.269	1.624	1.106	2.601
v_{46}	8.392	2.867	5.855	6.578	2.454	2.556	2.681	2.643	3.202	4.9	1.992	1.474	2.969
v_{47}	10.669	5.823	8.132	8.855	5.3	7.285	7.12	7.543	0.712	3.135	4.839	4.321	5.816
v_{48}	6.66	2.736	4.124	4.846	4.652	4.198	4.033	4.456	4.732	5.416	3.907	3.389	5.286
v_{49}	11.025	2.74	8.061	7.731	2.764	2.275	2.27	2.05	6.327	9.131	2.444	1.677	2.375
v_{50}	10.91	2.625	7.946	7.616	3.12	2.159	2.154	1.934	6.683	8.381	2.252	2.033	2.183
v_{51}	11.921	7.075	9.384	10.107	7.653	8.537	8.372	8.795	4.819	5.503	7.192	6.674	8.169
v_{52}	10.418	2.706	7.881	7.697	1.506	2.683	2.678	2.458	5.07	6.768	2.256	0.803	2.19
v_{53}	9.006	1.057	6.469	5.722	2.567	1.943	2.068	2.03	5.069	6.757	1.725	1.207	2.579
v_{54}	6.112	2.661	3.576	4.298	5.313	4.123	3.958	4.381	5.393	6.077	4.568	4.05	5.211
v_{55}	9.695	3.103	7.158	7.881	2.104	3.632	3.757	3.719	3.177	4.875	2.605	2.087	3.582
v_{56}	9.056	2.478	6.519	7.242	2.474	3.088	3.213	3.279	3.866	5.564	2.557	2.039	3.534
v_{57}	8.441	2.557	5.904	6.627	2.721	4.019	3.949	2.659	3.716	5.414	2.197	1.679	3.174
v_{58}	8.897	1.63	5.933	5.603	5.09	3.325	2.725	3.075	6.5	7.184	4.295	3.777	4.576
v_{59}	9.22	4.201	6.683	7.406	1.629	4.312	5.593	4.786	2.702	4.4	3.285	2.767	4.262
v_{60}	14.28	6.184	11.316	10.986	7.993	4.581	4.576	4.356	11.884	12.568	6.857	6.447	6.788
v_{61}	9.341	1.391	6.804	6.057	2.476	2.001	2.126	2.088	4.406	6.104	1.634	1.116	2.611
v_{62}	9.9	2.307	7.363	8.086	1.852	2.836	2.961	2.923	4.71	6.408	1.809	1.291	2.786
v_{63}	9.602	1.317	6.638	6.308	3.15	1.592	1.717	1.85	5.08	7.89	2.365	1.955	2.296
v_{64}	9.784	1.499	6.635	6.49	3.357	1.774	1.899	1.603	3.886	8.072	2.221	1.811	2.152
v_{65}	0	8.751	3.54	3.294	11.09	10.213	10.048	10.471	11.17	11.854	10.345	9.827	11.301
v_{66}	8.751	0	5.784	5.454	3.552	1.926	2.062	2.412	6.352	7.036	3.083	2.673	3.014
v_{67}	3.54	5.784	0	1.238	8.554	7.249	7.084	7.507	8.634	9.318	7.809	7.291	8.337
v_{68}	3.294	5.454	1.238	0	9.277	6.919	6.754	7.177	9.357	10.041	8.076	7.074	8.007
v_{69}	11.09	3.552	8.554	9.277	0	4.049	4.044	3.824	4.325	6.023	2.881	2.169	3.556
v_{70}	10.213	1.926	7.249	6.919	4.049	0	0.443	0.577	7.621	8.305	2.782	2.372	2.713
v_{71}	10.048	2.062	7.084	6.754	4.044	0.443	0	0.448	7.6	8.284	3.171	2.761	3.102
v_{72}	10.471	2.412	7.507	7.177	3.824	0.577	0.448	0	8.053	8.737	2.951	2.541	2.882
v_{73}	11.17	6.352	8.634	9.357	4.325	7.621	7.6	8.053	0	2.844	4.714	4.196	5.691
v_{74}	11.854	7.036	9.318	10.041	6.023	8.305	8.284	8.737	2.844	0	6.723	6.205	7.7
v_{75}	10.345	3.083	7.809	8.076	2.881	2.782	3.171	2.951	4.714	6.723	0	0.851	2.201
v_{76}	9.827	2.673	7.291	7.074	2.169	2.372	2.761	2.541	4.196	6.205	0.851	0	1.495
v_{77}	11.301	3.014	8.337	8.007	3.556	2.713	3.102	2.882	5.691	7.7	2.201	1.495	0
v_{78}	9.185	1.324	6.649	6.432	2.909	1.856	1.981	2.115	3.631	5.64	2.551	1.483	1.11
v_{79}	11.909	4.371	9.373	10.096	2.044	4.677	4.802	4.936	5.994	8.003	3.156	3.191	2.818
v_{80}	9.369	2.888	8.211	6.589	3.43	2.587	2.976	2.756	5.565	7.574	2.075	1.369	2.666
v_{81}	11.632	3.345	8.668	8.338	4.811	1.738	1.733	1.513	6.352	9.944	3.459	3.343	3.23
v_{82}	7.301	1.269	4.624	4.294	4.342	2.538	2.517	2.97	4.427	5.792	3.526	2.649	2.276
v_{83}	11.286	6.468	8.75	9.473	4.749	7.737	7.716	8.169	2.23	3.417	5.861	5.654	5.281
v_{84}	11.056	3.518	8.52	9.243	0.57	4.165	3.949	4.334	5.141	7.15	2.121	2.338	1.965
v_{85}	10.505	2.967	7.969	8.692	0.743	3.273	3.398	3.532	4.59	6.599	1.752	1.787	1.414
v_{86}	10.477	2.19	7.513	7.183	4.313	0.36	0.707	0.841	7.424	8.789	2.961	2.845	2.42
v_{87}	9.71	2.081	7.173	7.896	1.662	2.61	2.735	2.697	4.52	6.218	1.582	1.064	2.559
v_{88}	10.687	5.841	8.15	8.873	5.58	7.303	7.138	7.561	0.538	2.4	5.119	4.601	6.096
v_{89}	9.72	4.874	7.183	7.906	4.351	6.336	6.171	6.594	1.375	3.073	3.89	3.372	4.867
v_{90}	9.796	2.165	7.259	7.156	2.553	2.142	2.137	1.917	5.411	7.109	1.079	0.669	1.057
v_{91}	9.577	1.962	7.04	6.797	2.334	2.491	2.616	2.578	5.192	6.89	2.291	1.773	0.334
v_{92}	9.295	1.199	6.331	6.001	3.063	0.922	1.047	1.181	6.899	7.583	2.594	2.184	2.525
v_{93}	8.314	1.047	5.35	5.02	4.507	2.742	2.305	2.655	5.917	6.601	3.712	3.194	3.993
v_{94}	11.392	3.646	8.855	8.637	2.766	3.623	3.618	3.398	6.329	8.027	3.196	1.777	3.164
v_{95}	12.194	3.909	9.23	8.9	3.78	4.119	4.114	3.894	7.343	9.041	3.46	2.693	3.391
v_{96}	11.611	3.515	8.647	8.317	4.39	1.912	1.907	1.687	7.953	9.899	3.017	3.303	2.948
v_{97}	6.789	2.218	3.825	3.495	5.148	3.68	3.515	3.938	6.138	6.822	4.837	3.835	4.768
v_{98}	11.434	6.588	8.897	9.62	6.065	8.05	7.885	8.308	1.023	3.061	5.604	5.086	6.581
v_{99}	10.952	2.667	7.988	7.658	4.035	1.456	1.451	1.231	6.43	9.24	2.938	2.528	2.869
v_{100}	11.151	3.055	8.187	7.857	4.535	1.452	1.447	1.227	6.931	9.439	3.438	3.028	3.369
v_{101}	11.736	4.024	9.199	9.015	3.144	4.001	3.996	3.776	6.707	8.405	3.574	2.121	3.508
v_{102}	10.9	3.188	8.363	8.179	2.308	3.165	3.16	2.94	5.871	7.569	2.738	1.285	2.672

v_{103}	11.178	3.466	8.641	8.457	2.586	3.443	3.438	3.218	6.149	7.847	3.016	1.563	2.95
v_{104}	12.032	4.32	9.495	9.311	3.44	4.297	4.292	4.072	7.003	8.701	3.87	2.417	3.804
v_{105}	9.144	1.463	6.18	5.85	3.851	1.881	1.433	1.783	6.747	7.431	3.382	2.972	3.313
v_{106}	9.309	0.849	6.772	5.839	2.568	1.379	1.504	1.637	5.372	7.06	2.025	1.208	2.253
v_{107}	9.424	1.489	6.46	6.13	3.517	1.021	0.573	0.923	7.027	7.711	2.53	2.638	2.979
v_{108}	9.775	1.84	6.811	6.481	3.868	0.67	0.222	0.572	7.378	8.062	3.399	2.989	3.33
v_{109}	9.516	1.581	6.464	6.222	3.609	0.93	0.482	1.922	7.119	7.803	3.14	2.73	3.071
v_{110}	10.363	4.441	7.826	8.549	4.24	5.903	5.833	4.43	2.017	3.715	3.779	3.261	4.756
v_{111}	4.643	4.108	1.679	1.349	7.038	5.57	5.405	5.828	8.028	8.712	6.727	5.725	6.658
v_{112}	11.219	4.283	8.682	9.405	0.687	4.26	4.255	4.035	4.701	6.399	3.257	2.38	3.767
v_{113}	11.258	3.162	8.294	7.964	4.971	1.559	1.554	1.334	8.862	9.546	3.835	3.425	3.766
v_{114}	15.399	7.303	12.435	12.105	9.112	5.7	5.695	5.475	13.003	13.687	7.976	7.566	7.907
v_{115}	7.093	2.446	4.129	3.799	5.376	3.908	3.213	3.563	6.366	7.05	5.065	4.063	4.996
v_{116}	12.176	4.313	9.639	9.304	1.413	4.29	4.285	4.065	5.658	7.356	3.863	2.41	3.797
v_{117}	10.834	3.231	8.297	8.222	1.239	3.208	3.203	2.983	5.082	6.78	2.781	1.328	2.715
v_{118}	6.4	2.552	3.864	4.586	5.204	4.014	3.849	4.272	5.284	5.968	4.459	3.941	5.102
v_{119}	9.228	1.962	6.264	5.934	5.422	3.249	2.801	3.151	6.831	7.515	4.627	4.109	4.908
v_{120}	8.612	4.726	5.648	5.318	8.186	6.936	6.931	6.711	9.596	10.28	7.391	6.873	7.672
v_{121}	15.468	9.041	12.504	12.174	12.501	11.25	11.245	11.025	13.911	14.595	11.706	11.188	11.987
v_{122}	5.505	3.243	2.968	3.691	5.76	4.705	4.54	4.963	5.84	6.524	5.015	4.497	5.793
v_{123}	8.424	1.821	5.887	5.448	2.703	3.283	3.213	3.563	4.015	5.713	2.179	1.661	3.156
v_{124}	9.786	2.667	7.249	7.006	2.205	2.644	2.639	2.419	5.401	7.099	0.491	0.171	1.558
v_{125}	7.328	2.388	4.364	4.034	5.848	4.083	3.483	3.833	7.258	7.942	5.053	4.535	5.334
v_{126}	8.398	3.552	5.861	6.584	5.309	5.014	4.849	5.272	2.826	3.51	4.564	4.046	5.418
v_{127}	9.697	1.957	7.16	6.792	1.649	2.486	2.611	2.573	4.507	6.205	1.458	0.94	2.435
v_{128}	10.276	3.415	7.739	8.462	1.146	3.392	3.387	3.167	4.525	6.223	2.311	1.512	2.899
v_{129}	10.696	2.6	7.732	7.402	4.222	0.997	0.992	0.772	6.405	8.984	3.125	2.715	3.056
NODES	v_{78}	v_{79}	v_{80}	v_{81}	v_{82}	v_{83}	v_{84}	v_{85}	v_{86}	v_{87}	v_{88}	v_{89}	v_{90}
v_0	5.325	8.467	5.854	3.76	5.993	11.192	7.432	6.742	4.037	5.795	10.757	7.639	5.015
v_1	0.522	3.343	2.232	2.796	1.887	4.878	2.49	1.939	1.815	1.518	6.492	5.525	1.561
v_2	0.985	1.992	2.559	3.809	3.169	4.697	1.14	0.588	2.999	0.489	4.407	3.178	1.38
v_3	2.626	2.531	2.388	3.769	3.836	5.236	1.496	1.126	3.271	1.379	5.297	4.068	1.656
v_4	3.325	2.46	4.151	5.532	4.758	5.165	0.986	1.158	5.034	2.078	5.996	4.767	2.969
v_5	1.831	4.55	1.937	1.91	2.806	6.187	3.515	3.146	1.798	2.786	5.374	4.145	1.283
v_6	7.535	10.259	9.525	9.982	5.65	9.636	9.406	8.855	8.827	8.059	9.036	8.069	8.145
v_7	5.893	9.035	6.422	4.986	7.389	12.588	8	7.31	4.605	6.364	12.012	11.045	5.584
v_8	8.323	11.047	10.313	10.77	6.439	10.424	10.194	9.643	9.615	8.848	9.825	8.858	8.934
v_9	2.511	1.645	3.334	4.715	3.944	4.35	0.439	0.344	3.774	1.254	5.182	3.953	2.634
v_{10}	2.814	1.949	3.535	4.916	4.247	4.654	0.475	0.648	4.418	1.567	5.485	4.256	2.458
v_{11}	3.375	1.714	4.198	5.837	4.396	4.419	0.477	1.209	4.638	2.118	5.541	4.312	2.514
v_{12}	1.461	4.77	2.157	2.721	2.436	5.817	3.735	2.878	2.169	0.784	4.865	3.636	0.885
v_{13}	3.826	6.6	5.141	5.136	2.108	7.307	5.747	5.196	4.11	4.272	6.73	5.763	4.273
v_{14}	3.769	6.543	5.084	5.097	2.051	7.25	5.69	5.139	4.223	4.274	6.731	5.764	4.275
v_{15}	1.839	4.563	3.727	4.291	1.36	4.66	3.71	3.159	4.007	2.29	3.965	2.151	2.459
v_{16}	2.944	5.718	4.259	4.435	1.225	6.424	4.865	4.314	3.398	3.389	6.002	5.035	3.391
v_{17}	3.456	2.591	4.282	5.663	4.889	5.296	1.117	1.289	5.165	2.209	6.127	4.898	3.1
v_{18}	3.106	2.241	3.931	5.312	4.539	4.946	0.767	0.939	4.814	1.859	5.777	4.548	2.75
v_{19}	4.026	3.161	3.454	4.835	4.902	5.866	1.687	1.859	4.337	2.78	6.698	5.469	2.819
v_{20}	2.256	5.03	3.705	4.162	0.521	5.72	4.177	3.626	3.007	2.644	4.923	3.956	2.884
v_{21}	2.267	1.461	3.309	4.946	3.7	4.166	1.224	0.544	3.537	1.017	3.97	3.957	1.925
v_{22}	3.395	2.53	4.218	5.673	4.828	5.235	1.056	1.229	4.658	2.148	6.066	4.837	3.039
v_{23}	2.34	0.516	4.274	5.061	3.612	3.01	2.434	2.196	5.811	2.782	2.544	1.315	3.673
v_{24}	2.657	3.228	2.184	3.565	3.632	5.933	2.193	1.824	3.067	1.863	5.781	4.552	1.569
v_{25}	5.083	7.857	6.532	6.989	2.945	8.144	7.004	6.453	5.834	5.892	7.515	6.548	5.807
v_{26}	2.646	5.42	4.095	4.552	0.508	5.707	4.567	4.016	3.397	3.13	5.078	4.111	3.371
v_{27}	1.012	2.577	1.708	2.272	1.987	5.368	3.286	2.429	1.72	1.742	4.486	3.257	1.31
v_{28}	1.306	4.127	2.295	1.908	2.281	5.662	3.274	2.723	1.41	1.776	4.849	3.62	0.996
v_{29}	1.221	4.042	2.559	2.817	1.903	7.102	3.189	2.638	1.687	2.37	7.063	6.096	2.2
v_{30}	1.115	3.936	2.453	2.711	1.797	6.996	3.083	2.532	1.581	1.746	6.373	5.406	1.575

v_{31}	4.275	6.638	6.209	8.122	3.97	3.101	5.785	5.234	6.967	4.673	2.448	1.481	5.518
v_{32}	1.809	3.756	1.333	2.346	2.784	6.461	2.721	2.352	2.234	2.182	5.352	4.123	0.679
v_{33}	4.064	6.788	5.741	6.198	2.045	6.668	5.935	5.384	5.043	4.142	6.068	5.101	5.002
v_{34}	3.991	6.715	5.578	6.035	1.882	6.092	5.862	5.311	4.88	4.515	5.492	4.525	4.863
v_{35}	2.395	2.966	1.956	3.303	3.37	5.671	1.931	1.562	2.805	1.805	5.723	4.494	1.511
v_{36}	2.399	5.22	3.04	1.797	3.081	8.28	4.618	3.816	1.125	2.981	7.845	6.878	2.201
v_{37}	6.315	9.089	7.63	7.664	4.597	9.796	8.236	7.685	7.283	6.76	9.218	8.251	6.762
v_{38}	2.031	4.755	3.919	4.483	1.25	4.55	3.902	3.351	4.199	2.442	3.845	2.878	2.649
v_{39}	1.356	3.974	3.544	4.077	1.645	3.575	3.121	2.57	4.291	2.002	3.092	1.863	2.893
v_{40}	2.061	4.741	2.167	1.362	3.036	6.417	3.706	3.337	1.758	3.017	5.605	4.376	1.514
v_{41}	2.66	5.434	3.975	3.739	0.941	6.14	4.581	4.03	2.713	3.105	5.563	4.596	2.877
v_{42}	10.964	13.688	12.954	13.411	9.08	13.065	12.835	12.284	12.256	11.489	12.466	11.499	11.575
v_{43}	3.979	6.703	5.656	6.113	1.96	6.08	5.85	5.299	4.958	4.057	5.437	4.47	4.917
v_{44}	1.272	4.093	2.61	2.594	1.661	6.86	3.24	2.689	1.568	1.903	6.282	5.315	1.732
v_{45}	0.638	2.97	2.475	3.359	2.195	3.995	2.117	1.566	2.549	1.015	3.496	2.267	1.906
v_{46}	0.909	3.273	2.843	3.63	2.275	3.626	2.42	1.869	2.82	1.318	3.127	1.898	2.209
v_{47}	3.756	6.119	5.69	6.477	4.552	0.822	5.266	4.715	7.549	4.188	0.735	1.043	5.079
v_{48}	2.747	5.471	5.16	5.617	1.464	4.848	4.618	4.067	4.462	3.243	4.22	3.253	3.329
v_{49}	2.571	4.046	2.249	2.526	3.546	7.389	3.011	2.642	2.539	3.011	6.548	5.166	1.849
v_{50}	2.456	4.402	2.057	2.1	3.431	7.107	3.367	2.998	2.423	2.684	6.076	4.847	1.403
v_{51}	6.109	8.472	8.043	9.956	5.804	4.935	7.619	7.068	8.801	6.461	4.282	3.315	7.352
v_{52}	2.537	2.789	2.064	3.445	3.512	5.494	1.754	1.384	2.947	1.859	5.396	4.167	1.356
v_{53}	0.611	3.386	2.453	3.017	1.536	5.493	2.533	1.982	2.207	1.231	5.241	2.909	1.332
v_{54}	3.408	6.132	5.085	5.542	1.389	5.509	5.279	4.728	4.387	3.486	4.866	3.899	4.346
v_{55}	2.633	0.896	3.456	4.706	3.578	3.601	2.07	1.831	3.896	1.386	5.304	4.075	2.277
v_{56}	1.441	3.293	3.408	4.162	2.939	4.29	2.44	1.889	3.352	1.338	3.79	2.561	2.229
v_{57}	0.925	3.54	3.048	3.646	1.998	4.14	2.687	2.136	4.283	2.134	3.224	1.995	2.497
v_{58}	3.135	5.909	4.45	4.463	1.417	6.616	5.056	4.505	3.589	3.581	6.038	5.071	3.582
v_{59}	2.456	0.421	4.136	5.773	3.103	3.126	1.595	1.356	4.576	2.066	5.984	4.755	2.957
v_{60}	6.133	9.275	6.662	4.568	6.801	12	8.24	7.55	4.845	6.603	11.565	8.447	5.823
v_{61}	0.354	3.295	2.485	3.075	1.871	4.83	2.442	1.891	2.265	1.42	3.781	2.552	2.311
v_{62}	1.963	2.671	2.66	3.91	3.396	5.134	1.818	1.267	3.1	1.748	4.109	2.88	0.701
v_{63}	1.148	3.969	2.17	2.734	2.123	5.504	3.116	2.565	1.856	1.667	4.74	3.511	1.238
v_{64}	1.33	4.639	2.026	2.59	2.305	5.686	3.604	2.747	2.038	1.218	4.291	3.062	1.318
v_{65}	9.185	11.909	9.369	11.632	7.301	11.286	11.056	10.505	10.477	9.71	10.687	9.72	9.796
v_{66}	1.324	4.371	2.888	3.345	1.269	6.468	3.518	2.967	2.19	2.081	5.841	4.874	2.165
v_{67}	6.649	9.373	8.211	8.668	4.624	8.75	8.52	7.969	7.513	7.173	8.15	7.183	7.259
v_{68}	6.432	10.096	6.589	8.338	4.294	9.473	9.243	8.692	7.183	7.896	8.873	7.906	7.156
v_{69}	2.909	2.044	3.43	4.811	4.342	4.749	0.57	0.743	4.313	1.662	5.58	4.351	2.553
v_{70}	1.856	4.677	2.587	1.738	2.538	7.737	4.165	3.273	0.36	2.61	7.303	6.336	2.142
v_{71}	1.981	4.802	2.976	1.733	2.517	7.716	3.949	3.398	0.707	2.735	7.138	6.171	2.137
v_{72}	2.115	4.936	2.756	1.513	2.97	8.169	4.334	3.532	0.841	2.697	7.561	6.594	1.917
v_{73}	3.631	5.994	5.565	6.352	4.427	2.23	5.141	4.59	7.424	4.52	0.538	1.375	5.411
v_{74}	5.64	8.003	7.574	9.944	5.792	3.417	7.15	6.599	8.789	6.218	2.4	3.073	7.109
v_{75}	2.551	3.156	2.075	3.459	3.526	5.861	2.121	1.752	2.961	1.582	5.119	3.89	1.079
v_{76}	1.483	3.191	1.369	3.343	2.649	5.654	2.338	1.787	2.845	1.064	4.601	3.372	0.669
v_{77}	1.11	2.818	2.666	3.23	2.276	5.281	1.965	1.414	2.42	2.559	6.096	4.867	1.057
v_{78}	0	2.977	2.167	2.927	1.973	4.512	2.124	1.573	2.117	1.773	4.036	2.927	1.996
v_{79}	2.977	0	4.551	4.848	2.774	2.797	2.01	1.772	5.598	2.481	6.399	5.17	3.372
v_{80}	2.167	4.551	0	3.357	2.403	5.408	2.092	1.541	2.547	2.433	5.97	4.741	0.931
v_{81}	2.927	4.848	3.357	0	3.958	7.641	5.068	4.699	2.002	3.684	6.757	5.528	2.904
v_{82}	1.973	2.774	2.403	3.958	0	5.198	4.167	3.616	2.997	2.73	4.57	3.603	2.971
v_{83}	4.512	2.797	5.408	7.641	5.198	0	6.387	5.836	8.67	4.944	1.017	1.799	5.835
v_{84}	2.124	2.01	2.092	5.068	4.167	6.387	0	0.732	4.161	1.628	5.546	4.317	2.519
v_{85}	1.573	1.772	1.541	4.699	3.616	5.836	0.732	0	3.587	1.077	4.995	3.766	1.968
v_{86}	2.117	5.598	2.547	2.002	2.997	8.67	4.161	3.587	0	2.874	7.567	6.6	2.406
v_{87}	1.773	2.481	2.433	3.684	2.73	4.944	1.628	1.077	2.874	0	4.434	3.205	1.281
v_{88}	4.036	6.399	5.97	6.757	4.57	1.017	5.546	4.995	7.567	4.434	0	1.851	5.887
v_{89}	2.927	5.17	4.741	5.528	3.603	1.799	4.317	3.766	6.6	3.205	1.851	0	4.25
v_{90}	1.996	3.372	0.931	2.904	2.971	5.835	2.519	1.968	2.406	1.281	5.887	4.25	0

v_{91}	1.445	3.153	0.208	3.565	2.611	5.616	3.513	1.749	2.755	1.062	5.668	4.031	2.347
v_{92}	1.061	3.882	2.399	2.342	1.816	7.015	3.029	2.478	1.186	1.791	6.361	5.43	1.745
v_{93}	2.552	5.326	3.867	4.043	0.834	6.033	4.473	3.922	3.006	3.361	5.379	4.448	3.213
v_{94}	3.477	4.048	3.038	4.385	4.452	6.753	3.013	2.644	3.887	2.877	6.805	5.168	2.338
v_{95}	3.74	5.062	3.265	3.449	4.715	7.767	4.027	3.658	4.383	3.793	7.819	6.182	2.611
v_{96}	3.388	5.672	2.822	1.242	4.132	8.377	4.637	4.268	2.176	3.449	8.677	6.792	2.168
v_{97}	3.193	5.967	4.642	5.099	1.055	6.254	5.114	4.563	3.944	4.002	5.6	4.669	3.988
v_{98}	4.521	6.884	6.455	7.242	5.317	0.356	6.031	5.48	8.314	4.919	0.661	1.762	5.755
v_{99}	2.498	5.317	2.743	0.786	3.473	6.854	4.282	3.913	1.72	3.37	8.018	5.269	2.089
v_{100}	2.999	5.817	3.243	0.335	3.672	7.355	4.782	4.413	1.716	3.87	8.217	5.77	2.589
v_{101}	3.855	4.426	3.382	4.763	4.83	7.131	3.391	3.022	4.265	3.221	7.183	5.546	2.682
v_{102}	3.019	3.59	2.546	3.927	3.994	6.295	2.555	2.186	3.429	2.385	6.347	4.71	1.846
v_{103}	3.297	3.868	2.824	4.205	4.272	6.573	2.833	2.464	3.707	2.663	6.625	4.988	2.124
v_{104}	4.151	4.722	3.678	5.059	5.126	7.427	3.687	3.318	4.561	3.517	7.479	5.842	2.978
v_{105}	1.849	4.67	3.187	3.171	1.664	6.863	3.817	3.266	2.145	2.579	6.209	5.278	2.533
v_{106}	0.566	3.387	2.127	2.691	1.654	5.796	2.534	1.983	1.643	1.296	5.838	4.211	1.473
v_{107}	1.515	4.336	2.853	2.311	1.944	7.143	3.483	2.932	1.285	2.245	6.489	5.558	2.199
v_{108}	1.866	4.687	3.204	1.96	2.295	7.494	3.834	3.283	0.934	2.596	6.84	5.909	2.55
v_{109}	1.988	4.809	3.326	2.22	2.036	7.147	3.575	3.405	1.194	2.337	6.581	5.65	2.672
v_{110}	2.696	5.059	4.63	5.417	4.246	2.441	4.206	3.655	6.167	3.094	2.493	0.856	3.93
v_{111}	5.083	7.857	6.532	6.989	2.945	8.144	7.004	6.453	5.834	5.892	7.49	6.559	5.878
v_{112}	3.285	2.42	3.641	5.022	4.718	5.125	0.946	1.118	4.524	2.028	5.177	3.54	2.941
v_{113}	3.111	6.253	3.64	2.203	3.779	8.978	5.218	4.528	1.823	3.841	8.324	7.393	2.986
v_{114}	7.252	10.394	7.781	5.687	7.92	13.119	9.359	8.669	5.964	7.982	12.465	11.534	7.127
v_{115}	3.421	6.195	4.87	5.327	1.283	6.482	5.342	4.791	4.172	4.23	5.828	4.897	4.216
v_{116}	4.242	3.377	3.671	5.052	5.119	6.082	1.903	2.075	4.554	2.985	6.134	4.497	2.971
v_{117}	2.897	2.801	2.589	3.97	4.037	5.506	1.766	1.397	3.472	1.64	5.558	3.921	1.889
v_{118}	3.299	6.023	4.976	5.433	1.28	5.4	5.17	4.619	4.278	4.058	4.746	3.815	4.322
v_{119}	3.467	6.241	4.782	4.539	1.748	6.947	5.388	4.837	3.513	4.276	6.293	5.362	4.128
v_{120}	6.231	9.005	7.546	7.581	4.513	9.712	8.152	7.601	7.2	7.04	9.058	8.127	6.892
v_{121}	10.546	13.32	11.861	11.895	8.828	14.027	12.467	11.916	11.514	11.355	13.373	12.442	11.207
v_{122}	3.855	6.579	5.667	6.124	1.971	5.956	5.726	5.175	4.969	4.614	5.302	4.371	5.013
v_{123}	0.907	3.522	3.03	3.83	1.262	4.439	2.669	2.118	3.547	1.557	4.491	2.854	2.33
v_{124}	1.654	3.362	1.432	3.406	2.82	5.825	2.509	1.958	2.908	1.271	5.877	4.24	0.732
v_{125}	3.893	6.667	5.208	5.221	2.175	7.374	5.814	5.263	4.347	4.702	6.72	5.789	4.554
v_{126}	3.404	6.128	5.292	6.433	2.281	2.942	5.275	4.724	5.278	4.163	2.288	1.357	4.638
v_{127}	1.76	2.468	2.309	3.56	2.606	4.931	1.615	1.064	2.75	0.124	4.983	3.346	1.609
v_{128}	2.339	2.244	2.773	4.154	3.772	4.949	1.112	1.017	3.656	1.082	5.001	3.364	2.073
v_{129}	2.473	5.504	2.93	0.741	3.217	6.829	4.469	4.1	1.261	3.203	7.762	5.244	2.276
NODES	v_{91}	v_{92}	v_{93}	v_{94}	v_{95}	v_{96}	v_{97}	v_{98}	v_{99}	v_{100}	v_{101}	v_{102}	v_{103}
v_0	5.676	3.689	5.39	5.808	5.941	3.734	6.446	8.665	3.308	3.274	6.186	5.35	5.628
v_1	1.399	0.626	2.094	3.042	3.305	2.953	2.869	7.239	2.063	2.564	3.42	2.584	2.862
v_2	1.161	1.89	3.334	2.976	3.422	4.032	3.975	4.892	3.469	3.969	3.32	2.484	2.762
v_3	2.051	2.592	4.06	1.628	2.642	3.252	4.835	5.782	2.897	3.397	2.006	1.17	1.448
v_4	2.75	3.479	4.923	3.487	4.501	5.111	5.564	6.481	4.756	5.256	3.865	3.029	3.307
v_5	2.182	1.58	3.048	2.764	2.311	1.708	3.823	5.859	1.124	1.624	3.142	2.306	2.584
v_6	7.926	7.645	6.664	9.741	10.544	9.961	5.139	9.783	9.373	9.995	10.839	9.249	9.527
v_7	6.245	4.945	7.106	7.065	7.367	5.16	8.389	12.759	4.734	4.999	7.726	6.89	6.885
v_8	8.715	8.433	7.452	10.53	11.332	10.749	5.927	10.572	10.09	10.783	11.627	10.791	10.316
v_9	1.936	2.665	4.109	2.575	3.589	4.199	4.75	5.667	3.844	4.728	2.738	1.902	2.395
v_{10}	2.239	2.968	4.412	2.871	3.885	4.495	5.053	5.97	4.14	4.929	2.939	2.103	2.691
v_{11}	2.295	3.024	4.468	3.008	4.022	4.632	5.109	6.026	4.277	5.85	3.86	3.024	2.828
v_{12}	0.666	2.63	3.792	2.481	3.397	3.053	4.873	5.35	2.974	2.734	3.461	2.625	2.267
v_{13}	4.153	3.098	1.824	5.754	6.017	5.052	3.107	7.477	4.596	5.149	6.445	5.609	5.574
v_{14}	4.155	3.1	1.826	5.756	6.019	5.054	2.82	7.478	4.598	5.11	6.388	5.552	5.576
v_{15}	2.24	2.661	1.663	4.055	5.56	4.982	2.304	3.865	4.318	4.304	5.143	4.307	3.841
v_{16}	3.27	2.216	0.942	4.872	5.135	4.332	0.709	6.749	3.893	4.448	5.563	4.727	4.692
v_{17}	2.881	3.61	5.054	3.618	4.632	5.242	5.695	6.612	4.887	5.676	3.686	2.85	3.438
v_{18}	2.531	3.26	4.704	3.267	4.281	4.891	5.345	6.262	4.536	5.325	3.335	2.499	3.087

v_{19}	3.452	3.755	5.223	2.791	3.805	4.415	5.998	7.183	4.06	4.848	2.858	2.022	2.611
v_{20}	2.525	1.729	0.748	4.365	4.628	4.045	1.299	5.67	3.386	4.175	5.009	4.173	4.451
v_{21}	1.706	2.463	3.907	2.629	3.643	4.253	4.548	5.465	3.898	4.783	2.793	1.957	2.235
v_{22}	2.82	3.549	4.993	3.533	4.547	5.157	5.634	6.551	4.802	5.686	3.696	2.86	3.138
v_{23}	3.454	3.255	3.325	5.269	6.185	5.582	3.966	3.029	4.692	5.074	4.636	3.8	4.078
v_{24}	2.535	2.505	3.973	1.299	2.555	3.165	4.748	6.266	2.81	3.578	1.284	0.448	0.726
v_{25}	5.448	4.652	3.671	7.288	7.551	6.968	2.146	8.262	6.309	7.002	7.836	7	7.278
v_{26}	3.011	2.216	1.234	4.852	5.115	4.532	0.547	5.825	3.873	4.565	5.399	4.563	4.841
v_{27}	1.294	0.692	2.16	2.791	3.054	2.702	2.935	4.971	1.812	2.313	3.012	2.176	2.454
v_{28}	1.657	1.055	2.523	2.477	2.74	2.082	3.298	5.334	1.192	1.693	3.599	2.763	3.041
v_{29}	2.251	0.682	2.502	3.681	4.435	2.228	3.44	7.81	1.772	1.768	3.863	3.027	3.305
v_{30}	1.627	0.399	1.66	3.056	3.319	2.335	2.75	7.12	1.879	1.875	3.757	2.921	3.199
v_{31}	5.299	5.762	4.78	6.436	7.45	8.078	5.001	3.195	7.419	7.618	7.218	6.382	6.66
v_{32}	2.16	1.558	3.026	1.97	2.091	1.488	3.801	5.837	1.56	2.06	2.447	1.611	1.889
v_{33}	4.07	3.847	2.865	6.483	6.746	6.163	3.086	6.815	5.504	5.703	7.045	6.209	6.487
v_{34}	4.382	3.708	2.726	6.344	6.607	6.024	2.947	6.239	5.365	5.564	6.882	6.046	6.324
v_{35}	2.477	2.447	3.915	1.082	2.497	3.107	4.69	6.208	2.752	3.252	1.765	0.896	1.207
v_{36}	2.862	1.465	2.964	3.682	4.178	1.971	4.222	8.592	1.515	1.511	4.344	3.508	3.786
v_{37}	6.641	5.587	4.313	8.243	10.044	7.837	5.595	9.965	7.411	7.377	8.934	8.098	8.376
v_{38}	2.43	2.851	1.853	4.245	5.75	5.172	2.305	4.592	4.508	4.712	5.335	4.499	4.777
v_{39}	2.674	2.893	1.895	4.489	5.792	5.214	2.536	3.577	4.55	4.754	4.554	3.718	3.996
v_{40}	2.413	1.975	3.279	2.995	2.542	1.536	4.054	6.09	0.576	1.076	3.432	2.596	2.874
v_{41}	2.986	1.701	0.658	4.358	4.621	3.637	1.94	6.31	3.181	3.177	5.279	4.443	4.721
v_{42}	11.356	11.074	10.093	13.171	13.973	13.39	8.568	13.213	12.731	12.93	14.268	13.432	13.71
v_{43}	3.985	3.762	2.78	6.398	6.661	6.078	3.001	6.184	5.419	5.618	6.96	6.124	6.402
v_{44}	1.784	0.556	1.449	3.213	3.476	2.492	2.659	7.029	2.036	2.032	3.914	3.078	3.356
v_{45}	1.687	1.423	2.36	3.502	4.418	3.75	3.001	3.981	2.86	3.361	3.846	2.714	2.992
v_{46}	1.99	1.791	2.613	3.805	4.721	4.118	3.254	3.612	3.228	3.729	4.149	3.017	3.295
v_{47}	4.86	6.259	5.277	5.997	7.011	7.621	5.498	0.92	6.098	6.599	6.375	5.863	6.141
v_{48}	3.11	3.281	2.299	4.925	6.18	5.597	2.52	4.967	4.938	5.137	5.269	5.215	5.493
v_{49}	3.329	2.727	4.195	2.514	1.226	1.836	4.97	7.033	2.738	2.792	2.892	1.901	2.179
v_{50}	2.884	2.282	3.75	2.616	1.77	0.857	4.525	6.561	1.749	1.815	2.994	2.257	2.535
v_{51}	7.133	7.596	6.614	8.27	9.284	9.912	6.835	5.029	9.253	9.452	8.648	7.812	8.494
v_{52}	2.568	2.292	3.76	1.328	2.342	2.952	4.535	5.881	2.597	3.097	1.706	0.87	0.91
v_{53}	1.113	1.165	2.101	2.928	3.844	3.492	2.742	4.623	2.602	3.103	3.272	2.436	2.855
v_{54}	3.414	3.191	2.209	5.827	6.09	5.507	2.43	5.613	4.848	5.047	6.205	5.369	5.831
v_{55}	2.058	2.787	4.231	3.873	4.789	4.445	4.872	5.789	4.366	4.866	4.217	3.381	3.225
v_{56}	2.01	2.225	3.277	3.825	4.741	4.397	3.918	4.275	3.662	4.163	4.169	3.333	3.611
v_{57}	2.278	3.171	2.173	4.093	5.009	4.657	2.814	3.709	3.767	4.268	4.437	3.601	3.879
v_{58}	3.462	2.407	1.133	5.063	5.326	4.361	2.415	6.785	3.905	3.901	5.441	4.605	4.883
v_{59}	2.738	3.467	4.911	3.632	4.646	5.256	5.552	6.469	4.901	5.401	4.01	3.174	3.452
v_{60}	6.484	5.185	6.886	7.304	7.437	5.23	7.942	10.161	4.804	4.77	7.682	6.846	7.124
v_{61}	2.092	1.159	2.645	3.907	3.838	3.486	3.286	4.266	2.596	3.097	4.251	3.415	3.693
v_{62}	0.482	1.211	2.702	2.297	3.213	2.869	3.343	4.594	2.79	3.29	2.641	1.805	2.083
v_{63}	1.548	0.946	2.414	2.719	2.982	2.63	3.189	5.225	1.74	2.241	3.097	2.261	2.539
v_{64}	1.099	0.994	2.538	2.914	3.83	3.404	3.179	4.776	2.514	3.015	3.258	2.422	2.7
v_{65}	9.577	9.295	8.314	11.392	12.194	11.611	6.789	11.434	10.952	11.151	11.736	10.9	11.178
v_{66}	1.962	1.199	1.047	3.646	3.909	3.515	2.218	6.588	2.667	3.055	4.024	3.188	3.466
v_{67}	7.04	6.331	5.35	8.855	9.23	8.647	3.825	8.897	7.988	8.187	9.199	8.363	8.641
v_{68}	6.797	6.001	5.02	8.637	8.9	8.317	3.495	9.62	7.658	7.857	9.015	8.179	8.457
v_{69}	2.334	3.063	4.507	2.766	3.78	4.39	5.148	6.065	4.035	4.535	3.144	2.308	2.586
v_{70}	2.491	0.922	2.742	3.623	4.119	1.912	3.68	8.05	1.456	1.452	4.001	3.165	3.443
v_{71}	2.616	1.047	2.305	3.618	4.114	1.907	3.515	7.885	1.451	1.447	3.996	3.16	3.438
v_{72}	2.578	1.181	2.655	3.398	3.894	1.687	3.938	8.308	1.231	1.227	3.776	2.94	3.218
v_{73}	5.192	6.899	5.917	6.329	7.343	7.953	6.138	1.023	6.43	6.931	6.707	5.871	6.149
v_{74}	6.89	7.583	6.601	8.027	9.041	9.899	6.822	3.061	9.24	9.439	8.405	7.569	7.847
v_{75}	2.291	2.594	3.712	3.196	3.46	3.017	4.837	5.604	2.938	3.438	3.574	2.738	3.016
v_{76}	1.773	2.184	3.194	1.777	2.693	3.303	3.835	5.086	2.528	3.028	2.121	1.285	1.563
v_{77}	0.334	2.525	3.993	3.164	3.391	2.948	4.768	6.581	2.869	3.369	3.508	2.672	2.95
v_{78}	1.445	1.061	2.552	3.477	3.74	3.388	3.193	4.521	2.498	2.999	3.855	3.019	3.297

v_{79}	3.153	3.882	5.326	4.048	5.062	5.672	5.967	6.884	5.317	5.817	4.426	3.59	3.868
v_{80}	0.208	2.399	3.867	3.038	3.265	2.822	4.642	6.455	2.743	3.243	3.382	2.546	2.824
v_{81}	3.565	2.342	4.043	4.385	3.449	1.242	5.099	7.242	0.786	0.335	4.763	3.927	4.205
v_{82}	2.611	1.816	0.834	4.452	4.715	4.132	1.055	5.317	3.473	3.672	4.83	3.994	4.272
v_{83}	5.616	7.015	6.033	6.753	7.767	8.377	6.254	0.356	6.854	7.355	7.131	6.295	6.573
v_{84}	3.513	3.029	4.473	3.013	4.027	4.637	5.114	6.031	4.282	4.782	3.391	2.555	2.833
v_{85}	1.749	2.478	3.922	2.644	3.658	4.268	4.563	5.48	3.913	4.413	3.022	2.186	2.464
v_{86}	2.755	1.186	3.006	3.887	4.383	2.176	3.944	8.314	1.72	1.716	4.265	3.429	3.707
v_{87}	1.062	1.791	3.361	2.877	3.793	3.449	4.002	4.919	3.37	3.87	3.221	2.385	2.663
v_{88}	5.668	6.361	5.379	6.805	7.819	8.677	5.6	0.661	8.018	8.217	7.183	6.347	6.625
v_{89}	4.031	5.43	4.448	5.168	6.182	6.792	4.669	1.762	5.269	5.77	5.546	4.71	4.988
v_{90}	2.347	1.745	3.213	2.338	2.611	2.168	3.988	5.755	2.089	2.589	2.682	1.846	2.124
v_{91}	0	2.191	3.659	2.83	3.057	2.614	4.434	6.247	2.535	3.035	3.174	2.338	2.616
v_{92}	2.191	0	1.824	3.003	3.266	2.516	2.762	7.132	2.06	2.056	3.381	2.545	2.823
v_{93}	3.659	1.824	0	4.48	4.743	3.941	1.832	6.202	3.501	3.481	4.858	4.022	4.3
v_{94}	2.83	3.003	4.48	0	3.58	4.19	5.773	7.291	3.835	4.335	1.72	0.851	1.162
v_{95}	3.057	3.266	4.743	3.58	0	1.838	5.363	7.426	2.739	2.794	3.285	2.449	2.727
v_{96}	2.614	2.516	3.941	4.19	1.838	0	5.273	8.405	0.901	0.956	4.264	3.428	3.706
v_{97}	4.434	2.762	1.832	5.773	5.363	5.273	0	6.372	4.419	4.618	5.776	4.94	5.218
v_{98}	6.247	7.132	6.202	7.291	7.426	8.405	6.372	0	8.679	8.878	7.295	6.459	6.737
v_{99}	2.535	2.06	3.501	3.835	2.739	0.901	4.419	8.679	0	0.5	3.948	3.112	3.39
v_{100}	3.035	2.056	3.481	4.335	2.794	0.956	4.618	8.878	0.5	0	4.776	3.94	4.218
v_{101}	3.174	3.381	4.858	1.72	3.285	4.264	5.776	7.295	3.948	4.776	0	0.868	0.605
v_{102}	2.338	2.545	4.022	0.851	2.449	3.428	4.94	6.459	3.112	3.94	0.868	0	0.31
v_{103}	2.616	2.823	4.3	1.162	2.727	3.706	5.218	6.737	3.39	4.218	0.605	0.31	0
v_{104}	3.47	3.677	5.154	2.015	3.581	4.56	6.072	7.591	4.244	5.072	0.294	1.163	0.887
v_{105}	2.979	1.133	1.381	4.318	3.908	3.345	2.611	6.87	2.889	3.184	4.491	3.655	3.933
v_{106}	1.919	0.461	2.107	3.258	2.848	2.865	2.6	6.499	1.975	2.704	3.431	2.595	2.873
v_{107}	2.645	0.799	1.732	3.984	3.574	2.485	2.891	7.15	2.029	2.324	4.157	3.321	3.599
v_{108}	2.996	1.274	2.083	4.335	3.972	2.134	3.242	7.501	1.678	1.973	4.508	3.672	3.95
v_{109}	2.737	1.272	1.665	4.076	4.232	2.394	2.895	7.242	1.938	2.233	4.63	3.413	3.691
v_{110}	4.422	4.985	4.15	5.466	5.601	6.58	5.301	2.605	4.701	5.43	5.639	4.803	5.081
v_{111}	6.324	4.652	3.722	7.663	7.253	7.163	2.146	8.151	6.38	7.002	7.836	7	7.278
v_{112}	3.433	3.64	4.986	2.872	3.544	4.523	5.665	5.289	4.207	5.035	3.045	2.209	2.487
v_{113}	3.432	2.163	3.588	4.771	4.215	2.377	4.725	8.985	1.951	2.216	4.944	4.108	4.386
v_{114}	7.573	6.304	7.729	8.912	8.187	6.349	8.866	13.126	5.923	6.188	9.085	8.249	8.527
v_{115}	4.662	2.99	1.458	6.001	5.591	5.501	0.56	6.489	4.718	5.34	6.174	5.338	5.616
v_{116}	3.463	3.67	5.147	2.902	3.574	4.553	6.065	6.246	4.237	5.065	3.075	2.239	2.517
v_{117}	2.381	2.588	4.065	1.82	2.492	3.471	4.983	5.67	3.155	3.983	1.993	1.157	1.435
v_{118}	4.768	3.096	2.166	6.107	5.697	5.607	1.227	5.407	4.824	5.446	6.28	5.444	5.722
v_{119}	4.574	2.501	1.465	5.913	5.503	4.713	2.695	6.954	4.257	4.552	6.086	5.25	5.528
v_{120}	7.338	5.503	4.229	8.677	9.593	7.755	5.699	9.719	7.329	7.594	8.85	8.014	8.292
v_{121}	11.653	9.818	8.544	12.992	13.907	12.069	9.775	14.034	11.643	11.908	13.165	12.329	12.607
v_{122}	5.459	3.787	2.857	6.986	6.388	6.298	3.027	5.963	5.515	6.137	7.159	6.323	6.601
v_{123}	2.822	2.365	1.53	3.929	3.987	4.004	2.209	4.603	3.114	3.843	4.102	3.266	3.544
v_{124}	1.224	2.024	3.088	2.005	2.312	3.291	3.767	5.989	2.591	3.419	2.178	1.342	1.62
v_{125}	5	3.165	1.891	6.339	5.929	5.395	3.122	7.381	4.939	5.234	6.512	5.676	5.954
v_{126}	5.084	4.096	3.166	6.535	6.013	6.607	3.336	2.949	5.14	6.446	6.708	5.872	6.15
v_{127}	2.101	1.568	2.874	2.875	3.28	4.259	3.553	5.095	2.844	3.573	3.048	2.212	2.49
v_{128}	2.565	2.772	4.04	2.004	2.676	3.655	4.719	5.113	3.339	4.167	2.177	1.341	1.619
v_{129}	2.722	1.601	3.026	4.022	2.753	0.915	4.163	8.423	0.459	0.754	4.195	3.359	3.637
NODES	v_{104}	v_{105}	v_{106}	v_{107}	v_{108}	v_{109}	v_{110}	v_{111}	v_{112}	v_{113}	v_{114}	v_{115}	v_{116}
v_0	6.482	4.518	4.145	3.658	3.307	3.567	6.84	8.336	7.133	3.881	2.791	6.674	6.475
v_1	3.716	1.414	0.354	1.08	1.431	1.172	3.251	4.759	3.679	2.676	6.817	3.097	3.709
v_2	3.616	2.678	1.395	2.344	2.695	2.436	3.067	5.865	1.549	3.94	8.081	4.203	2.506
v_3	2.302	3.38	2.32	3.046	3.397	3.138	3.957	6.725	1.433	3.833	7.974	5.063	1.463
v_4	4.161	4.267	2.984	3.933	4.284	4.025	4.656	7.454	0.51	5.692	9.833	5.792	1.468
v_5	3.438	2.368	1.308	2.065	1.714	1.974	4.034	5.713	3.401	2.193	6.334	4.051	3.431
v_6	10.381	7.494	7.658	7.774	8.125	7.778	8.712	2.993	9.568	9.608	13.749	5.443	10.525

v_7	7.739	5.774	5.401	4.914	4.563	4.823	10.706	9.88	7.702	2.81	5.347	7.653	7.732
v_8	11.17	8.282	8.447	8.562	8.913	8.566	9.501	3.781	10.357	10.396	14.537	6.231	11.314
v_9	3.249	3.453	2.17	3.119	3.47	3.592	3.842	6.64	0.848	4.78	8.921	4.978	1.805
v_{10}	3.545	3.756	2.473	3.422	3.773	3.895	4.145	6.943	0.592	5.076	9.217	5.281	1.518
v_{11}	3.682	3.812	2.529	3.478	3.829	3.57	4.201	6.999	0.941	5.213	9.354	5.337	1.898
v_{12}	3.121	3.418	2.358	3.084	3.435	3.176	3.525	6.763	2.459	3.871	8.012	5.101	3.114
v_{13}	6.428	2.131	3.381	2.843	3.194	2.935	5.424	4.299	6.26	4.915	7.97	2.37	6.421
v_{14}	6.43	2.133	3.383	2.845	3.196	2.937	5.426	2.33	6.262	4.701	9.759	2.372	6.423
v_{15}	4.695	2.494	1.972	2.773	3.124	2.778	2.04	4.194	3.799	4.629	8.77	2.532	4.756
v_{16}	5.546	1.411	2.498	2.123	2.474	2.215	4.541	2.599	5.377	3.979	8.12	0.517	5.539
v_{17}	4.292	4.398	3.115	4.064	4.415	4.156	4.787	7.585	0.641	5.823	9.964	5.923	1.599
v_{18}	3.941	4.048	2.765	3.714	4.065	3.806	4.437	7.235	0.29	5.472	9.613	5.573	1.248
v_{19}	3.465	4.543	3.483	4.209	4.56	4.301	5.358	7.888	0.742	4.996	9.137	6.226	0.217
v_{20}	5.303	1.726	1.842	1.858	2.209	1.95	3.796	3.189	4.632	3.692	7.833	1.527	5.032
v_{21}	3.087	3.178	2.178	2.917	3.268	3.009	3.64	6.438	1.604	4.834	8.975	4.776	2.561
v_{22}	3.99	4.264	3.264	4.003	4.354	4.095	4.726	7.524	0.819	5.738	9.879	5.862	1.777
v_{23}	4.93	4.624	2.857	4.435	4.786	4.527	1.204	5.856	4.294	5.305	9.446	4.194	5.251
v_{24}	1.578	2.97	2.131	2.959	3.31	3.051	4.441	6.638	1.847	3.746	7.887	4.976	1.877
v_{25}	8.13	4.553	4.669	4.781	5.132	4.873	7.191	0	7.555	6.615	10.756	2.45	7.955
v_{26}	5.693	2.116	2.232	2.344	2.695	2.436	4.754	2.437	5.118	4.179	8.32	0.775	5.519
v_{27}	3.306	1.325	0.486	1.146	1.497	1.619	3.146	4.825	3.087	2.426	6.566	3.163	3.458
v_{28}	3.893	1.619	0.78	1.677	1.326	1.586	3.509	5.188	3.114	1.806	5.946	3.526	3.144
v_{29}	4.157	0.84	0.611	0.506	0.857	1.95	5.663	5.33	4.318	1.875	6.016	3.668	4.348
v_{30}	4.051	0.734	0.505	0.4	0.751	0.492	4.973	4.64	3.693	1.982	6.123	2.978	3.723
v_{31}	7.512	5.686	5.802	5.965	6.316	6.057	2.124	6.891	4.808	7.725	11.866	5.229	5.765
v_{32}	2.741	2.122	1.283	1.788	2.192	1.88	4.012	5.691	2.607	2.629	6.77	4.029	2.637
v_{33}	7.339	3.762	3.878	4.041	4.392	4.133	5.744	5.933	5.654	5.81	9.951	3.314	6.611
v_{34}	7.176	3.599	3.715	3.878	4.229	3.97	5.168	5.357	6.024	5.671	9.812	3.175	6.981
v_{35}	2.059	2.708	1.869	2.374	2.725	2.466	4.383	6.58	1.789	3.688	7.829	4.918	1.819
v_{36}	4.638	2.018	1.789	1.223	0.872	1.132	4.714	6.112	4.319	1.618	5.759	4.45	4.349
v_{37}	9.228	4.602	5.767	5.314	5.665	5.071	7.912	6.595	8.748	5.487	8.542	4.859	8.91
v_{38}	5.629	3.012	3.034	3.292	3.643	3.384	2.192	4.195	3.951	4.819	8.96	2.533	4.908
v_{39}	4.848	3.104	3.126	3.384	3.735	3.476	1.752	4.426	3.511	4.861	9.002	2.764	4.468
v_{40}	3.726	2.374	1.535	2.067	1.716	2.513	4.265	5.944	3.632	1.959	5.931	4.282	3.662
v_{41}	5.573	0.716	2.112	1.428	1.779	1.52	4.257	3.83	5.093	3.284	7.425	1.204	5.025
v_{42}	14.562	10.975	11.091	11.254	11.605	11.367	12.142	6.422	12.998	13.037	17.178	8.872	13.955
v_{43}	7.254	3.677	3.793	3.956	4.307	4.048	5.113	4.891	5.569	5.725	9.866	3.229	6.526
v_{44}	4.208	0.577	0.662	0.283	0.634	0.375	4.977	4.549	3.85	2.139	6.28	2.357	3.88
v_{45}	3.844	3.293	1.155	1.821	2.172	1.913	2.156	4.891	2.527	3.473	7.614	3.229	3.484
v_{46}	4.147	3.406	1.426	2.092	2.443	3.799	1.787	5.144	2.83	3.841	7.982	3.482	3.787
v_{47}	6.993	6.268	6.384	6.547	6.898	6.66	1.685	7.388	4.369	8.222	12.363	5.726	5.326
v_{48}	6.345	3.181	3.297	3.46	3.811	3.552	3.896	4.41	4.752	5.244	9.385	2.748	5.709
v_{49}	3.031	2.884	2.045	2.848	2.497	3.023	5.208	6.86	3.151	4.213	8.185	5.198	3.181
v_{50}	3.387	2.769	1.93	2.732	2.381	2.908	4.736	6.415	3.253	2.818	6.959	4.753	3.283
v_{51}	9.346	7.52	7.636	7.799	8.15	7.912	3.958	8.725	6.642	9.559	13.7	7.063	7.599
v_{52}	1.762	2.85	2.011	2.516	2.867	2.989	4.056	6.425	1.965	3.533	7.674	4.763	1.995
v_{53}	3.707	2.634	0.813	1.479	1.83	3.027	2.798	4.632	2.906	3.215	7.356	2.97	3.561
v_{54}	6.683	3.106	3.222	3.385	3.736	3.477	4.542	4.32	4.998	5.154	9.295	2.658	5.955
v_{55}	4.077	3.502	2.502	3.168	3.519	3.641	3.964	6.762	2.48	4.837	8.978	5.1	3.437
v_{56}	4.465	4.108	1.73	2.679	3.03	2.771	2.45	5.808	2.85	4.275	8.416	4.146	3.807
v_{57}	4.733	3.004	2.482	3.283	3.634	3.375	1.884	4.704	3.643	5.139	9.28	3.042	4.6
v_{58}	5.737	1.44	2.69	2.152	2.503	2.244	4.733	3.608	5.569	5.607	8.662	1.679	5.73
v_{59}	4.306	4.255	2.972	3.921	4.272	4.013	4.644	7.442	2.005	5.837	9.978	5.78	2.962
v_{60}	7.978	6.014	5.641	5.154	4.803	5.063	8.336	9.832	7.941	4.689	1.119	8.17	7.971
v_{61}	4.547	1.947	0.664	1.613	1.964	1.705	2.441	5.176	2.932	3.209	7.35	3.514	3.889
v_{62}	2.937	1.999	0.716	1.665	2.016	1.757	2.769	5.233	2.9	3.261	7.402	3.571	2.93
v_{63}	3.393	1.734	0.674	1.4	1.751	1.492	3.4	5.079	3.356	2.354	6.494	3.417	3.386
v_{64}	3.554	1.781	0.552	1.447	1.798	1.539	2.951	5.069	2.892	3.127	7.268	3.407	3.547
v_{65}	12.032	9.144	9.309	9.424	9.775	9.516	10.363	4.643	11.219	11.258	15.399	7.093	12.176
v_{66}	4.32	1.463	0.849	1.489	1.84	1.581	4.441	4.108	4.283	3.162	7.303	2.446	4.313

v_{67}	9.495	6.18	6.772	6.46	6.811	6.464	7.826	1.679	8.682	8.294	12.435	4.129	9.639
v_{68}	9.311	5.85	5.839	6.13	6.481	6.222	8.549	1.349	9.405	7.964	12.105	3.799	9.304
v_{69}	3.44	3.851	2.568	3.517	3.868	3.609	4.24	7.038	0.687	4.971	9.112	5.376	1.413
v_{70}	4.297	1.881	1.379	1.021	0.67	0.93	5.903	5.57	4.26	1.559	5.7	3.908	4.29
v_{71}	4.292	1.433	1.504	0.573	0.222	0.482	5.833	5.405	4.255	1.554	5.695	3.213	4.285
v_{72}	4.072	1.783	1.637	0.923	0.572	1.922	4.43	5.828	4.035	1.334	5.475	3.563	4.065
v_{73}	7.003	6.747	5.372	7.027	7.378	7.119	2.017	8.028	4.701	8.862	13.003	6.366	5.658
v_{74}	8.701	7.431	7.06	7.711	8.062	7.803	3.715	8.712	6.399	9.546	13.687	7.05	7.356
v_{75}	3.87	3.382	2.025	2.53	3.399	3.14	3.779	6.727	3.257	3.835	7.976	5.065	3.863
v_{76}	2.417	2.972	1.208	2.638	2.989	2.73	3.261	5.725	2.38	3.425	7.566	4.063	2.41
v_{77}	3.804	3.313	2.253	2.979	3.33	3.071	4.756	6.658	3.767	3.766	7.907	4.996	3.797
v_{78}	4.151	1.849	0.566	1.515	1.866	1.988	2.696	5.083	3.285	3.111	7.252	3.421	4.242
v_{79}	4.722	4.67	3.387	4.336	4.687	4.809	5.059	7.857	2.42	6.253	10.394	6.195	3.377
v_{80}	3.678	3.187	2.127	2.853	3.204	3.326	4.63	6.532	3.641	3.64	7.781	4.87	3.671
v_{81}	5.059	3.171	2.691	2.311	1.96	2.22	5.417	6.989	5.022	2.203	5.687	5.327	5.052
v_{82}	5.126	1.664	1.654	1.944	2.295	2.036	4.246	2.945	4.718	3.779	7.92	1.283	5.119
v_{83}	7.427	6.863	5.796	7.143	7.494	7.147	2.441	8.144	5.125	8.978	13.119	6.482	6.082
v_{84}	3.687	3.817	2.534	3.483	3.834	3.575	4.206	7.004	0.946	5.218	9.359	5.342	1.903
v_{85}	3.318	3.266	1.983	2.932	3.283	3.405	3.655	6.453	1.118	4.528	8.669	4.791	2.075
v_{86}	4.561	2.145	1.643	1.285	0.934	1.194	6.167	5.834	4.524	1.823	5.964	4.172	4.554
v_{87}	3.517	2.579	1.296	2.245	2.596	2.337	3.094	5.892	2.028	3.841	7.982	4.23	2.985
v_{88}	7.479	6.209	5.838	6.489	6.84	6.581	2.493	7.49	5.177	8.324	12.465	5.828	6.134
v_{89}	5.842	5.278	4.211	5.558	5.909	5.65	0.856	6.559	3.54	7.393	11.534	4.897	4.497
v_{90}	2.978	2.533	1.473	2.199	2.55	2.672	3.93	5.878	2.941	2.986	7.127	4.216	2.971
v_{91}	3.47	2.979	1.919	2.645	2.996	2.737	4.422	6.324	3.433	3.432	7.573	4.662	3.463
v_{92}	3.677	1.133	0.461	0.799	1.274	1.272	4.985	4.652	3.64	2.163	6.304	2.99	3.67
v_{93}	5.154	1.381	2.107	1.732	2.083	1.665	4.15	3.722	4.986	3.588	7.729	1.458	5.147
v_{94}	2.015	4.318	3.258	3.984	4.335	4.076	5.466	7.663	2.872	4.771	8.912	6.001	2.902
v_{95}	3.581	3.908	2.848	3.574	3.972	4.232	5.601	7.253	3.544	4.215	8.187	5.591	3.574
v_{96}	4.56	3.345	2.865	2.485	2.134	2.394	6.58	7.163	4.523	2.377	6.349	5.501	4.553
v_{97}	6.072	2.611	2.6	2.891	3.242	2.895	5.301	2.146	5.665	4.725	8.866	0.56	6.065
v_{98}	7.591	6.87	6.499	7.15	7.501	7.242	2.605	8.151	5.289	8.985	13.126	6.489	6.246
v_{99}	4.244	2.889	1.975	2.029	1.678	1.938	4.701	6.38	4.207	1.951	5.923	4.718	4.237
v_{100}	5.072	3.184	2.704	2.324	1.973	2.233	5.43	7.002	5.035	2.216	6.188	5.34	5.065
v_{101}	0.294	4.491	3.431	4.157	4.508	4.63	5.639	7.836	3.045	4.944	9.085	6.174	3.075
v_{102}	1.163	3.655	2.595	3.321	3.672	3.413	4.803	7	2.209	4.108	8.249	5.338	2.239
v_{103}	0.887	3.933	2.873	3.599	3.95	3.691	5.081	7.278	2.487	4.386	8.527	5.616	2.517
v_{104}	0	4.785	3.725	4.451	4.802	4.543	5.933	8.13	3.339	5.238	9.379	6.468	3.369
v_{105}	4.785	0	1.249	0.86	1.211	0.952	4.98	4.553	4.427	2.716	6.857	1.927	4.457
v_{106}	3.725	1.249	0	0.905	1.256	0.997	3.213	4.669	3.588	2.585	6.726	3.007	3.618
v_{107}	4.451	0.86	0.905	0	0.351	0.092	5.26	4.832	4.093	2.132	6.273	2.64	4.123
v_{108}	4.802	1.211	1.256	0.351	0	0.259	5.611	5.183	4.444	1.781	5.922	2.991	4.474
v_{109}	4.543	0.952	0.997	0.092	0.259	0	5.352	4.924	4.185	2.041	6.182	2.732	4.215
v_{110}	5.933	4.98	3.213	5.26	5.611	5.352	0	6.212	3.319	5.661	9.802	4.55	4.276
v_{111}	8.13	4.553	4.669	4.832	5.183	4.924	6.212	0	7.555	6.615	10.756	2.45	7.955
v_{112}	3.339	4.427	3.588	4.093	4.444	4.185	3.319	7.555	0	5.182	9.323	5.752	0.958
v_{113}	5.238	2.716	2.585	2.132	1.781	2.041	5.661	6.615	5.182	0	5.351	5.148	4.95
v_{114}	9.379	6.857	6.726	6.273	5.922	6.182	9.802	10.756	9.323	5.351	0	9.289	9.09
v_{115}	6.468	1.927	3.007	2.64	2.991	2.732	4.55	2.45	5.752	5.148	9.289	0	6.294
v_{116}	3.369	4.457	3.618	4.123	4.474	4.215	4.276	7.955	0.958	4.95	9.09	6.294	0
v_{117}	2.287	3.375	2.536	3.041	3.392	3.133	3.7	6.873	1.45	3.868	8.008	5.212	1.48
v_{118}	6.574	2.997	3.113	3.276	3.627	3.368	3.818	3.117	5.58	5.254	9.395	1.456	6.538
v_{119}	6.38	1.516	2.919	2.228	2.579	2.32	4.596	4.585	5.798	5.635	8.69	2.012	6.354
v_{120}	9.144	4.518	5.683	5.23	5.581	5.322	7.36	3.97	8.562	5.405	8.46	5.895	9.118
v_{121}	13.459	8.833	9.998	9.545	9.896	9.637	11.675	10.826	12.877	9.719	12.774	9.091	13.433
v_{122}	7.453	3.688	3.804	3.967	4.318	4.059	5.047	5.041	6.136	5.945	10.086	3.256	7.094
v_{123}	4.396	2.36	2.382	2.64	2.991	2.732	2.633	4.099	3.079	4.523	8.664	2.438	4.037
v_{124}	2.472	2.811	1.972	2.477	2.828	2.569	4.019	5.657	2.416	3.304	7.444	3.996	2.446
v_{125}	6.806	2.198	3.345	2.91	3.261	2.667	5.022	2.685	6.224	6.861	9.916	2.438	6.78
v_{126}	7.002	3.997	4.113	4.276	4.627	4.368	2.033	5.226	5.685	6.254	10.395	3.565	6.643

v_{127}	3.342	2.356	1.356	2.022	2.373	2.114	3.125	5.443	2.025	3.458	7.598	3.782	2.983
v_{128}	2.471	3.559	2.72	3.225	3.576	3.698	3.143	6.609	1.521	4.052	8.192	4.948	1.944
v_{129}	4.489	2.154	1.947	1.57	1.219	1.479	5.023	6.053	4.433	1.462	5.434	4.392	4.463
NODES	v_{117}	v_{118}	v_{119}	v_{120}	v_{121}	v_{122}	v_{123}	v_{124}	v_{125}	v_{126}	v_{127}	v_{128}	v_{129}
v_0	5.393	6.78	6.074	5.844	10.158	7.471	6.049	4.829	7.3	7.78	4.983	5.577	2.819
v_1	2.627	3.203	3.009	5.773	10.088	3.894	1.662	2.063	3.435	4.203	1.394	2.811	2.038
v_2	1.161	4.031	4.249	7.013	11.328	4.587	1.53	1.37	4.675	4.136	0.476	0.603	3.302
v_3	0.381	4.921	4.975	7.739	12.054	5.477	2.42	1.067	5.401	5.026	1.366	0.496	3.084
v_4	1.96	5.62	5.838	8.602	12.917	6.176	3.119	2.926	6.264	5.725	2.065	1.561	4.943
v_5	2.349	4.157	3.963	7.57	11.884	4.848	2.447	1.785	4.389	4.473	2.662	2.533	1.311
v_6	9.183	4.749	7.578	6.962	13.818	3.854	6.773	8.135	5.678	6.747	8.046	8.625	9.046
v_7	6.65	8.723	6.149	5.919	10.233	9.414	8.086	6.086	7.375	9.723	6.24	6.834	4.245
v_8	9.972	5.538	8.366	7.75	14.606	4.643	7.562	8.924	6.466	7.536	8.835	9.414	9.834
v_9	1.328	4.806	5.024	7.788	12.103	5.362	2.305	2.014	5.45	4.911	1.251	0.674	4.031
v_{10}	1.344	5.109	5.327	8.091	12.406	5.665	2.608	2.31	5.753	5.214	1.554	1.051	4.327
v_{11}	1.761	5.165	5.383	8.147	12.462	5.721	2.664	2.504	5.809	5.27	1.61	1.107	4.464
v_{12}	2.032	4.489	4.707	7.471	11.786	5.045	1.988	0.875	5.133	4.594	0.66	1.513	3.161
v_{13}	5.339	3.441	0.867	2.553	6.868	4.132	2.804	4.362	2.093	4.441	4.148	5.314	4.137
v_{14}	5.341	3.442	1.808	3.671	8.656	4.133	2.806	4.364	0.356	4.442	4.15	5.316	4.139
v_{15}	3.414	1.571	2.578	5.342	9.657	2.127	1.087	2.449	3.004	1.676	2.277	2.856	4.067
v_{16}	4.457	1.604	1.495	4.259	8.574	3.404	1.921	3.479	1.921	3.713	3.265	4.431	3.417
v_{17}	2.091	5.751	5.969	8.733	13.048	6.307	3.25	3.057	6.395	5.856	2.196	1.692	5.074
v_{18}	1.74	5.401	5.619	8.383	12.698	5.957	2.9	2.706	6.045	5.506	1.846	1.342	4.723
v_{19}	1.264	6.322	6.138	8.902	13.217	6.878	3.821	2.23	6.564	6.427	2.767	1.728	4.247
v_{20}	3.95	1.634	1.662	4.427	8.742	2.325	1.176	2.734	2.089	2.634	2.52	3.686	3.13
v_{21}	1.382	4.604	4.822	7.586	11.901	5.16	2.103	1.943	5.248	4.709	1.049	0.825	4.085
v_{22}	2.269	5.69	5.908	8.672	12.987	6.246	3.189	3.029	6.334	5.795	2.135	1.632	4.989
v_{23}	3.906	3.79	4.24	7.004	11.319	4.346	2.277	3.663	4.666	1.331	2.769	3.348	4.667
v_{24}	0.795	5.082	4.888	7.652	11.967	5.961	2.904	0.98	5.314	5.51	1.85	0.979	2.997
v_{25}	6.873	3.117	4.585	3.97	10.826	5.041	4.099	5.657	2.685	5.226	5.443	6.609	6.053
v_{26}	4.437	0.764	2.148	5.989	9.228	2.48	1.662	3.22	2.575	2.789	3.006	4.172	3.617
v_{27}	2.376	3.269	3.075	5.839	10.154	3.96	1.559	1.503	3.501	3.585	1.289	2.56	1.787
v_{28}	2.062	3.632	3.438	7.183	11.497	4.323	1.922	1.498	3.864	3.948	1.652	2.246	1.167
v_{29}	3.266	3.774	3.179	7.252	11.566	4.465	3.043	2.702	3.843	4.774	2.246	3.45	1.313
v_{30}	2.641	3.084	2.102	5.104	9.419	3.775	2.353	2.077	2.784	4.084	1.622	2.825	1.42
v_{31}	5.189	4.147	5.694	8.459	12.774	4.703	4.354	5.508	6.121	1.689	4.614	4.632	7.163
v_{32}	1.555	4.135	3.941	6.705	11.02	4.826	2.425	1.181	4.367	4.451	2.058	1.739	1.747
v_{33}	5.266	1.12	3.779	6.544	10.859	1.067	3.293	4.279	4.206	3.779	4.129	4.708	5.248
v_{34}	5.639	0.981	3.64	6.405	10.72	0.491	3.229	4.591	4.067	3.203	4.502	5.081	5.109
v_{35}	0.737	5.024	4.83	7.594	11.909	5.903	2.846	0.922	5.256	5.452	1.792	0.921	2.939
v_{36}	3.267	4.556	3.46	6.995	11.309	5.247	3.825	2.703	4.142	5.556	2.857	3.451	1.056
v_{37}	7.828	5.929	3.356	2.634	4.231	6.62	5.292	6.85	4.582	6.929	6.636	7.802	6.922
v_{38}	3.566	1.451	2.768	5.532	9.847	2.007	1.277	2.639	3.194	1.556	2.429	3.008	4.257
v_{39}	3.126	1.978	2.81	5.574	9.889	2.534	1.163	2.883	3.236	2.083	1.989	2.568	4.299
v_{40}	2.58	4.388	4.295	7.337	11.651	5.079	2.678	2.016	4.62	4.704	2.893	2.764	0.763
v_{41}	3.943	2.274	0.8	3.802	8.117	2.965	1.637	3.195	1.475	3.274	2.981	4.147	2.722
v_{42}	12.613	8.179	11.007	10.391	17.247	7.284	10.203	11.565	9.107	10.177	11.476	12.055	12.475
v_{43}	5.181	1.035	3.694	6.459	10.774	1.37	3.208	4.194	4.121	3.148	4.044	4.623	5.163
v_{44}	2.798	2.993	1.945	4.947	9.262	3.684	2.357	2.234	2.627	3.993	1.779	2.982	1.577
v_{45}	2.139	3.119	3.275	6.039	10.354	3.675	0.556	1.896	3.701	3.224	1.002	1.581	2.835
v_{46}	2.442	2.75	3.528	6.292	10.607	3.306	0.669	2.199	3.954	2.855	1.305	1.884	3.203
v_{47}	4.75	4.644	6.191	8.956	13.271	5.2	3.683	5.069	6.618	2.186	4.175	4.193	6.073
v_{48}	4.367	0.63	3.213	5.978	10.293	1.32	1.957	3.319	3.64	1.931	3.23	3.809	4.682
v_{49}	2.099	5.304	5.11	9.591	13.905	5.995	3.594	1.919	5.536	5.62	2.887	2.283	2.751
v_{50}	2.201	4.859	4.665	8.195	12.509	5.55	3.149	2.021	5.091	5.175	2.782	2.385	1.774
v_{51}	7.023	5.981	7.528	10.293	14.608	6.537	6.188	7.342	7.955	3.523	6.448	6.466	8.997
v_{52}	0.913	4.869	4.675	7.439	11.754	5.292	2.456	0.767	5.101	4.841	1.735	1.097	2.784
v_{53}	2.479	2.847	3.016	5.78	10.095	3.403	0.927	1.322	3.442	2.952	1.107	1.96	2.577
v_{54}	4.61	0.464	3.123	5.888	10.203	0.798	2.637	3.623	3.55	2.577	3.473	4.052	4.592

v_{55}	2.372	4.928	5.146	7.91	12.225	5.484	2.427	2.267	5.572	5.033	1.373	1.814	4.199
v_{56}	2.462	3.414	4.192	6.956	11.271	3.97	1.235	2.219	4.618	3.519	1.325	1.904	3.637
v_{57}	3.258	2.31	3.088	5.852	10.167	2.866	1.125	2.487	3.514	2.415	2.121	2.7	3.742
v_{58}	4.648	2.749	0.71	3.244	7.559	3.44	2.113	3.671	1.402	3.749	3.457	4.623	3.446
v_{59}	2.385	5.608	5.826	8.59	12.905	6.164	3.107	2.947	6.252	5.713	2.053	1.828	5.088
v_{60}	6.889	8.276	7.571	7.341	11.655	8.967	7.545	6.325	8.797	9.276	6.479	7.073	4.315
v_{61}	2.544	3.404	3.56	6.324	10.639	3.96	0.841	2.301	3.986	3.509	1.407	1.986	2.571
v_{62}	1.848	3.449	3.617	6.381	10.696	4.005	1.169	0.691	4.043	3.554	1.735	2.032	2.623
v_{63}	2.304	3.523	3.329	6.093	10.408	4.214	1.813	1.74	3.755	3.839	1.543	2.488	1.715
v_{64}	2.465	3.285	3.453	6.217	10.532	3.841	1.364	1.308	3.879	3.39	1.094	1.946	2.489
v_{65}	10.834	6.4	9.228	8.612	15.468	5.505	8.424	9.786	7.328	8.398	9.697	10.276	10.696
v_{66}	3.231	2.552	1.962	4.726	9.041	3.243	1.821	2.667	2.388	3.552	1.957	3.415	2.6
v_{67}	8.297	3.864	6.264	5.648	12.504	2.968	5.887	7.249	4.364	5.861	7.16	7.739	7.732
v_{68}	8.222	4.586	5.934	5.318	12.174	3.691	5.448	7.006	4.034	6.584	6.792	8.462	7.402
v_{69}	1.239	5.204	5.422	8.186	12.501	5.76	2.703	2.205	5.848	5.309	1.649	1.146	4.222
v_{70}	3.208	4.014	3.249	6.936	11.25	4.705	3.283	2.644	4.083	5.014	2.486	3.392	0.997
v_{71}	3.203	3.849	2.801	6.931	11.245	4.54	3.213	2.639	3.483	4.849	2.611	3.387	0.992
v_{72}	2.983	4.272	3.151	6.711	11.025	4.963	3.563	2.419	3.833	5.272	2.573	3.167	0.772
v_{73}	5.082	5.284	6.831	9.596	13.911	5.84	4.015	5.401	7.258	2.826	4.507	4.525	6.405
v_{74}	6.78	5.968	7.515	10.28	14.595	6.524	5.713	7.099	7.942	3.51	6.205	6.223	8.984
v_{75}	2.781	4.459	4.627	7.391	11.706	5.015	2.179	0.491	5.053	4.564	1.458	2.311	3.125
v_{76}	1.328	3.941	4.109	6.873	11.188	4.497	1.661	0.171	4.535	4.046	0.94	1.512	2.715
v_{77}	2.715	5.102	4.908	7.672	11.987	5.793	3.156	1.558	5.334	5.418	2.435	2.899	3.056
v_{78}	2.897	3.299	3.467	6.231	10.546	3.855	0.907	1.654	3.893	3.404	1.76	2.339	2.473
v_{79}	2.801	6.023	6.241	9.005	13.32	6.579	3.522	3.362	6.667	6.128	2.468	2.244	5.504
v_{80}	2.589	4.976	4.782	7.546	11.861	5.667	3.03	1.432	5.208	5.292	2.309	2.773	2.93
v_{81}	3.97	5.433	4.539	7.581	11.895	6.124	3.83	3.406	5.221	6.433	3.56	4.154	0.741
v_{82}	4.037	1.28	1.748	4.513	8.828	1.971	1.262	2.82	2.175	2.281	2.606	3.772	3.217
v_{83}	5.506	5.4	6.947	9.712	14.027	5.956	4.439	5.825	7.374	2.942	4.931	4.949	6.829
v_{84}	1.766	5.17	5.388	8.152	12.467	5.726	2.669	2.509	5.814	5.275	1.615	1.112	4.469
v_{85}	1.397	4.619	4.837	7.601	11.916	5.175	2.118	1.958	5.263	4.724	1.064	1.017	4.1
v_{86}	3.472	4.278	3.513	7.2	11.514	4.969	3.547	2.908	4.347	5.278	2.75	3.656	1.261
v_{87}	1.64	4.058	4.276	7.04	11.355	4.614	1.557	1.271	4.702	4.163	0.124	1.082	3.203
v_{88}	5.558	4.746	6.293	9.058	13.373	5.302	4.491	5.877	6.72	2.288	4.983	5.001	7.762
v_{89}	3.921	3.815	5.362	8.127	12.442	4.371	2.854	4.24	5.789	1.357	3.346	3.364	5.244
v_{90}	1.889	4.322	4.128	6.892	11.207	5.013	2.33	0.732	4.554	4.638	1.609	2.073	2.276
v_{91}	2.381	4.768	4.574	7.338	11.653	5.459	2.822	1.224	5	5.084	2.101	2.565	2.722
v_{92}	2.588	3.096	2.501	5.503	9.818	3.787	2.365	2.024	3.165	4.096	1.568	2.772	1.601
v_{93}	4.065	2.166	1.465	4.229	8.544	2.857	1.53	3.088	1.891	3.166	2.874	4.04	3.026
v_{94}	1.82	6.107	5.913	8.677	12.992	6.986	3.929	2.005	6.339	6.535	2.875	2.004	4.022
v_{95}	2.492	5.697	5.503	9.593	13.907	6.388	3.987	2.312	5.929	6.013	3.28	2.676	2.753
v_{96}	3.471	5.607	4.713	7.755	12.069	6.298	4.004	3.291	5.395	6.607	4.259	3.655	0.915
v_{97}	4.983	1.227	2.695	5.699	9.775	3.027	2.209	3.767	3.122	3.336	3.553	4.719	4.163
v_{98}	5.67	5.407	6.954	9.719	14.034	5.963	4.603	5.989	7.381	2.949	5.095	5.113	8.423
v_{99}	3.155	4.824	4.257	7.329	11.643	5.515	3.114	2.591	4.939	5.14	2.844	3.339	0.459
v_{100}	3.983	5.446	4.552	7.594	11.908	6.137	3.843	3.419	5.234	6.446	3.573	4.167	0.754
v_{101}	1.993	6.28	6.086	8.85	13.165	7.159	4.102	2.178	6.512	6.708	3.048	2.177	4.195
v_{102}	1.157	5.444	5.25	8.014	12.329	6.323	3.266	1.342	5.676	5.872	2.212	1.341	3.359
v_{103}	1.435	5.722	5.528	8.292	12.607	6.601	3.544	1.62	5.954	6.15	2.49	1.619	3.637
v_{104}	2.287	6.574	6.38	9.144	13.459	7.453	4.396	2.472	6.806	7.002	3.342	2.471	4.489
v_{105}	3.375	2.997	1.516	4.518	8.833	3.688	2.36	2.811	2.198	3.997	2.356	3.559	2.154
v_{106}	2.536	3.113	2.919	5.683	9.998	3.804	2.382	1.972	3.345	4.113	1.356	2.72	1.947
v_{107}	3.041	3.276	2.228	5.23	9.545	3.967	2.64	2.477	2.91	4.276	2.022	3.225	1.57
v_{108}	3.392	3.627	2.579	5.581	9.896	4.318	2.991	2.828	3.261	4.627	2.373	3.576	1.219
v_{109}	3.133	3.368	2.32	5.322	9.637	4.059	2.732	2.569	2.667	4.368	2.114	3.698	1.479
v_{110}	3.7	3.818	4.596	7.36	11.675	5.047	2.633	4.019	5.022	2.033	3.125	3.143	5.023
v_{111}	6.873	3.117	4.585	3.97	10.826	5.041	4.099	5.657	2.685	5.226	5.443	6.609	6.053
v_{112}	1.45	5.58	5.798	8.562	12.877	6.136	3.079	2.416	6.224	5.685	2.025	1.521	4.433
v_{113}	3.868	5.254	5.635	5.405	9.719	5.945	4.523	3.304	6.861	6.254	3.458	4.052	1.462
v_{114}	8.008	9.395	8.69	8.46	12.774	10.086	8.664	7.444	9.916	10.395	7.598	8.192	5.434

v_{115}	5.212	1.456	2.012	5.895	9.091	3.256	2.438	3.996	2.438	3.565	3.782	4.948	4.392
v_{116}	1.48	6.538	6.354	9.118	13.433	7.094	4.037	2.446	6.78	6.643	2.983	1.944	4.463
v_{117}	0	5.192	5.176	7.94	12.255	5.748	2.691	1.268	5.602	5.297	1.637	0.766	3.285
v_{118}	5.192	0	3.028	6.669	10.108	1.059	2.542	3.528	3.455	2.482	3.378	3.957	4.497
v_{119}	5.176	3.028	0	3.272	7.587	3.772	2.444	4.002	1.964	4.081	3.788	4.954	3.522
v_{120}	7.94	6.669	3.272	0	6.865	9.01	5.209	6.767	4.007	6.845	6.553	7.719	6.839
v_{121}	12.255	10.108	7.587	6.865	0	10.851	9.523	11.081	8.813	11.16	10.867	12.033	11.153
v_{122}	5.748	1.059	3.772	9.01	10.851	0	3.093	4.455	4.156	3.067	4.366	4.945	5.198
v_{123}	2.691	2.542	2.444	5.209	9.523	3.093	0	1.624	3.145	2.656	1.409	2.262	2.879
v_{124}	1.268	3.528	4.002	6.767	11.081	4.455	1.624	0	4.706	4.217	1.111	1.485	2.778
v_{125}	5.602	3.455	1.964	4.007	8.813	4.156	3.145	4.706	0	4.566	4.274	5.44	4.263
v_{126}	5.297	2.482	4.081	6.845	11.16	3.067	2.656	4.217	4.566	0	3.939	4.473	5.475
v_{127}	1.637	3.378	3.788	6.553	10.867	4.366	1.409	1.111	4.274	3.939	0	1.069	3.19
v_{128}	0.766	3.957	4.954	7.719	12.033	4.945	2.262	1.485	5.44	4.473	1.069	0	3.47
v_{129}	3.285	4.497	3.522	6.839	11.153	5.198	2.879	2.778	4.263	5.475	3.19	3.47	0

Appendix C

Table of Demands per Barangay

Table C.1: Demands per Barangay

Node	Demand (m^3)	Node	Demand (m^3)	Node	Demand (m^3)
v_0	0	v_{44}	4.5785	v_{88}	2.9035
v_1	9.9896	v_{45}	5.4237	v_{89}	11.884
v_2	3.1035	v_{46}	9.2017	v_{90}	9.8903
v_3	11.706	v_{47}	9.1568	v_{91}	5.1186
v_4	2.8667	v_{48}	3.3872	v_{92}	2.6103
v_5	3.0061	v_{49}	11.394	v_{93}	5.181
v_6	8.0353	v_{50}	7.7676	v_{94}	2.5386
v_7	4.6618	v_{51}	7.9318	v_{95}	8.5012
v_8	8.0986	v_{52}	10.11	v_{96}	9.8115
v_9	8.5557	v_{53}	9.8442	v_{97}	7.9099
v_{10}	3.9625	v_{54}	9.6259	v_{98}	6.9795
v_{11}	2.7735	v_{55}	10.962	v_{99}	5.9629
v_{12}	5.968	v_{56}	9.138	v_{100}	5.0435
v_{13}	9.5811	v_{57}	4.4623	v_{101}	3.4251
v_{14}	6.5259	v_{58}	9.2235	v_{102}	10.803
v_{15}	5.1752	v_{59}	7.399	v_{103}	9.8485
v_{16}	3.0592	v_{60}	5.3068	v_{104}	9.3496
v_{17}	7.1656	v_{61}	10.186	v_{105}	11.15
v_{18}	7.2504	v_{62}	7.3189	v_{106}	9.8045
v_{19}	11.326	v_{63}	7.567	v_{107}	11.636
v_{20}	9.6733	v_{64}	9.7555	v_{108}	3.6157
v_{21}	10.523	v_{65}	6.5938	v_{109}	10.03
v_{22}	4.2417	v_{66}	7.7162	v_{110}	7.4813
v_{23}	10.416	v_{67}	8.1739	v_{111}	11.79
v_{24}	8.4012	v_{68}	9.6668	v_{112}	6.1863
v_{25}	10.006	v_{69}	2.8296	v_{113}	9.424
v_{26}	4.2771	v_{70}	10.397	v_{114}	10.549
v_{27}	2.1818	v_{71}	6.4385	v_{115}	5.2415
v_{28}	2.4182	v_{72}	6.294	v_{116}	4.3412
v_{29}	9.6683	v_{73}	5.9388	v_{117}	10.103
v_{30}	7.8353	v_{74}	11.011	v_{118}	11.805
v_{31}	10.865	v_{75}	2.2611	v_{119}	2.0034
v_{32}	10.432	v_{76}	4.1811	v_{120}	10.323
v_{33}	6.7749	v_{77}	11.091	v_{121}	3.2554
v_{34}	6.8976	v_{78}	5.3653	v_{122}	3.296
v_{35}	4.1015	v_{79}	3.5416	v_{123}	7.022
v_{36}	10.116	v_{80}	8.9962	v_{124}	11.621
v_{37}	11.36	v_{81}	8.4553	v_{125}	7.7346
v_{38}	11.791	v_{82}	4.5679	v_{126}	8.1474
v_{39}	5.2775	v_{83}	11.715	v_{127}	3.0507
v_{40}	8.2457	v_{84}	3.1612	v_{128}	6.293
v_{41}	9.3105	v_{85}	5.9067	v_{129}	7.2159
v_{42}	5.5314	v_{86}	7.7326		
v_{43}	8.2028	v_{87}	11.711		

Appendix D

Complete Results for 4 Nodes

Table D.1: Complete Results for 4 Nodes

Population Size 5							
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time (s)	Converged	Successful
PS X 5	1	1337.798049	1337.798049	1337.798049	11.698318	1	1
	2	1337.798049	2316.808305	1533.600100	14.644609	0	0
	3	1337.798049	1337.798049	1337.798049	8.692495	1	1
	4	1358.105721	1358.105721	1358.105721	1.891671	1	0
	5	1337.798049	1358.105721	1345.921181	14.627273	0	0
	6	1337.798049	1337.798049	1337.798049	14.641393	0	0
	7	1337.798049	1337.798049	1337.798049	10.526195	1	1
	8	1337.798049	1908.770406	1552.178792	14.618708	0	0
	9	1337.798049	1337.798049	1337.798049	14.124159	1	1
	10	1337.798049	1337.798049	1337.798049	9.294622	1	1
	Total	13398.288162	14968.578447	13816.594088	114.759443	6	5
	Mean	1339.828816	1496.857845	1381.659409	11.475944		
	STD. Dev	6.42184975	338.6730857	85.33446882	4.112270315		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 10	1	1337.798049	1337.798049	1337.798049	6.284802	1	1
	2	1337.798049	1337.798049	1337.798049	18.644605	1	1
	3	1337.798049	1337.798049	1337.798049	13.069274	1	1
	4	1337.798049	1337.798049	1337.798049	20.457753	1	1
	5	1337.798049	1337.798049	1337.798049	13.802123	1	1
	6	1337.798049	1337.798049	1337.798049	4.980491	1	1
	7	1337.798049	1337.798049	1337.798049	10.483271	1	1
	8	1337.798049	1337.798049	1337.798049	19.033980	1	1
	9	1337.798049	1337.798049	1337.798049	6.201007	1	1
	10	1337.798049	1337.798049	1337.798049	7.487129	1	1
	Total	13377.980490	13377.980490	13377.980490	120.444435	10	10
	Mean	1337.798049	1337.798049	1337.798049	12.044444		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	5.847737113		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 15	1	1337.798049	1337.798049	1337.798049	7.422273	1	1
	2	1337.798049	1337.798049	1337.798049	18.946113	1	1
	3	1337.798049	1337.798049	1337.798049	10.437414	1	1
	4	1337.798049	1337.798049	1337.798049	19.544259	1	1
	5	1358.105721	1358.105721	1358.105721	3.740614	1	0
	6	1337.798049	1337.798049	1337.798049	9.218090	1	1
	7	1337.798049	1337.798049	1337.798049	15.918786	1	1
	8	1337.798049	1337.798049	1337.798049	11.673607	1	1
	9	1337.798049	1337.798049	1337.798049	11.643156	1	1
	10	1337.798049	1337.798049	1337.798049	14.090555	1	1
	Total	13398.288162	13398.288162	13398.288162	122.634867	10	9
	Mean	1339.828816	1339.828816	1339.828816	12.263487		
	STD. Dev	6.42184975	6.42184975	6.42184975	4.986481626		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 20	1	1337.798049	1337.798049	1337.798049	3.783788	1	1

	2	1337.798049	1337.798049	1337.798049	34.230664	1	1
	3	1337.798049	1337.798049	1337.798049	17.756826	1	1
	4	1337.798049	1337.798049	1337.798049	13.520821	1	1
	5	1337.798049	1337.798049	1337.798049	10.476216	1	1
	6	1337.798049	1337.798049	1337.798049	8.644347	1	1
	7	1337.798049	1337.798049	1337.798049	36.030153	1	1
	8	1337.798049	1337.798049	1337.798049	15.970190	1	1
	9	1337.798049	1337.798049	1337.798049	7.431154	1	1
	10	1337.798049	1337.798049	1337.798049	2.560630	1	1
	Total	13377.980490	13377.980490	13377.980490	150.404789	10	10
	Mean	1337.798049	1337.798049	1337.798049	15.040479		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	11.64992337		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 25	1	1337.798049	1337.798049	1337.798049	14.153978	1	1
	2	1337.798049	1337.798049	1337.798049	34.255901	1	1
	3	1337.798049	1337.798049	1337.798049	29.336058	1	1
	4	1358.105721	1358.105721	1358.105721	3.795271	1	0
	5	1337.798049	1337.798049	1337.798049	22.714296	1	1
	6	1337.798049	1337.798049	1337.798049	6.257190	1	1
	7	1337.798049	1337.798049	1337.798049	12.973958	1	1
	8	1337.798049	1337.798049	1337.798049	12.406948	1	1
	9	1337.798049	1337.798049	1337.798049	20.257527	1	1
	10	1337.798049	1337.798049	1337.798049	22.714086	1	1
	Total	13398.288162	13398.288162	13398.288162	178.865213	10	9
	Mean	1339.828816	1339.828816	1339.828816	17.886521		
	STD. Dev	6.42184975	6.42184975	6.42184975	9.736406493		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 30	1	1337.798049	1337.798049	1337.798049	20.894310	1	1
	2	1337.798049	1337.798049	1337.798049	16.034722	1	1
	3	1337.798049	1337.798049	1337.798049	6.865592	1	1
	4	1337.798049	1337.798049	1337.798049	33.092359	1	1
	5	1337.798049	1337.798049	1337.798049	42.821620	1	1
	6	1337.798049	1337.798049	1337.798049	10.530092	1	1
	7	1337.798049	1337.798049	1337.798049	20.264431	1	1
	8	1337.798049	1337.798049	1337.798049	37.945243	1	1
	9	1337.798049	1337.798049	1337.798049	29.406300	1	1
	10	1337.798049	1337.798049	1337.798049	18.535820	1	1
	Total	13377.980490	13377.980490	13377.980490	236.390489	10	10
	Mean	1337.798049	1337.798049	1337.798049	23.639049		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	11.79099684		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 35	1	1337.798049	1337.798049	1337.798049	6.945964	1	1
	2	1337.798049	1337.798049	1337.798049	19.722528	1	1
	3	1337.798049	1337.798049	1337.798049	5.688550	1	1
	4	1337.798049	1337.798049	1337.798049	11.781271	1	1
	5	1337.798049	1337.798049	1337.798049	9.330526	1	1
	6	1337.798049	1337.798049	1337.798049	10.563466	1	1
	7	1337.798049	1337.798049	1337.798049	6.289345	1	1
	8	1337.798049	1337.798049	1337.798049	27.006376	1	1
	9	1337.798049	1337.798049	1337.798049	28.856660	1	1
	10	1337.798049	1337.798049	1337.798049	24.590302	1	1
	Total	13377.980490	13377.980490	13377.980490	150.774988	10	10
	Mean	1337.798049	1337.798049	1337.798049	15.077499		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	9.0676978		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	1337.798049	1337.798049	15.807147	1	1
	2	1337.798049	1337.798049	1337.798049	14.226266	1	1
	3	1358.105721	1358.105721	1358.105721	4.474146	1	0

PS X 40

	4	1337.798049	1337.798049	1337.798049	20.313746	1	1
	5	1337.798049	1337.798049	1337.798049	1.979463	1	1
	6	1337.798049	1337.798049	1337.798049	13.603500	1	1
	7	1337.798049	1337.798049	1337.798049	74.654883	1	1
	8	1337.798049	1337.798049	1337.798049	10.574218	1	1
	9	1337.798049	1337.798049	1337.798049	11.187458	1	1
	10	1337.798049	1337.798049	1337.798049	22.805738	1	1
	Total	13398.288162	13398.288162	13398.288162	189.626565	10	9
	Mean	1339.828816	1339.828816	1339.828816	18.962657		
	STD. Dev	6.42184975	6.42184975	6.42184975	20.57100282		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 45	1	1337.798049	1337.798049	1337.798049	19.013494	1	1
	2	1337.798049	1337.798049	1337.798049	17.330960	1	1
	3	1337.798049	1337.798049	1337.798049	9.973465	1	1
	4	1337.798049	1337.798049	1337.798049	62.385824	1	1
	5	1337.798049	1337.798049	1337.798049	37.408583	1	1
	6	1337.798049	1337.798049	1337.798049	34.369151	1	1
	7	1337.798049	1337.798049	1337.798049	3.236624	1	1
	8	1337.798049	1337.798049	1337.798049	5.330725	1	1
	9	1337.798049	1337.798049	1337.798049	50.241411	1	1
	10	1838.729406	1838.729406	1838.729406	1.300687	1	0
	Total	13878.911847	13878.911847	13878.911847	240.590924	10	9
	Mean	1387.891185	1387.891185	1387.891185	24.059092		
	STD. Dev	158.408404	158.408404	158.408404	21.10913717		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 50	1	1337.798049	1337.798049	1337.798049	21.010583	1	1
	2	1337.798049	1337.798049	1337.798049	25.280921	1	1
	3	1337.798049	1337.798049	1337.798049	12.447649	1	1
	4	1337.798049	1337.798049	1337.798049	19.780430	1	1
	5	1337.798049	1337.798049	1337.798049	31.973812	1	1
	6	1337.798049	1337.798049	1337.798049	6.943062	1	1
	7	1337.798049	1337.798049	1337.798049	1.306488	1	1
	8	1337.798049	1337.798049	1337.798049	5.138219	1	1
	9	1337.798049	1337.798049	1337.798049	4.521536	1	1
	10	1337.798049	1337.798049	1337.798049	10.011842	1	1
	Total	13377.980490	13377.980490	13377.980490	138.414542	10	10
	Mean	1337.798049	1337.798049	1337.798049	13.841454		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	10.16661548		

Population Size 10

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 5	1	1337.798049	2475.047972	1549.424067	31.562551	0	0
	2	1337.798049	1337.798049	1337.798049	21.872496	1	1
	3	1337.798049	2545.088972	1558.458934	31.019887	0	0
	4	1337.798049	1337.798049	1337.798049	19.384370	1	1
	5	1337.798049	1337.798049	1337.798049	27.973919	1	1
	6	1337.798049	1337.798049	1337.798049	14.388359	1	1
	7	1337.798049	1337.798049	1337.798049	23.716507	1	1
	8	1337.798049	2475.047972	1555.516369	30.977077	0	0
	9	1337.798049	1586.386380	1364.687650	31.049659	0	0
	10	1337.798049	1337.798049	1337.798049	26.801331	1	1
	Total	13377.980490	17108.359590	14054.875314	258.746155	6	6
	Mean	1337.798049	1710.835959	1405.487531	25.874616		
	STD. Dev	2.39673E-13	549.1862144	103.1624689	5.884445212		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	1337.798049	1337.798049	18.164296	1	1
	2	1337.798049	1337.798049	1337.798049	41.171017	1	1
	3	1337.798049	1586.386388	1389.546484	61.584723	0	0
	4	1337.798049	1337.798049	1337.798049	26.879632	1	1

PS X 10

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	5	1337.798049	1358.105721	1341.859837	61.605850	0	0
	6	1337.798049	1337.798049	1337.798049	57.507035	1	1
	7	1337.798049	1337.798049	1337.798049	30.630701	1	1
	8	1337.798049	1337.798049	1337.798049	54.678383	1	1
	9	1337.798049	1337.798049	1337.798049	43.583823	1	1
	10	1337.798049	1778.571073	1381.875352	61.643988	0	0
	Total	13377.980490	14087.649525	13477.868016	457.449448	7	7
	Mean	1337.798049	1408.764952	1347.786802	45.744945		
	STD. Dev	2.39673E-13	151.3371542	20.10929995	16.13837174		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 15	1	1337.798049	1337.798049	1337.798049	30.726615	1	1
	2	1337.798049	1337.798049	1337.798049	41.268981	1	1
	3	1337.798049	1337.798049	1337.798049	8.322709	1	1
	4	1337.798049	1337.798049	1337.798049	25.095402	1	1
	5	1337.798049	1337.798049	1337.798049	85.764006	1	1
	6	1337.798049	1566.078716	1383.454183	92.200099	0	0
	7	1337.798049	1337.798049	1337.798049	49.897258	1	1
	8	1337.798049	1337.798049	1337.798049	67.906231	1	1
	9	1337.798049	1337.798049	1337.798049	43.056086	1	1
	10	1337.798049	1337.798049	1337.798049	78.440160	1	1
	Total	13377.980490	13606.261157	13377.980490	522.677546	9	9
	Mean	1337.798049	1360.626116	1337.798049	52.267755		
	STD. Dev	2.39673E-13	72.18868535	2.39673E-13	27.87097112		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 20	1	1337.798049	1337.798049	1337.798049	91.028405	1	1
	2	1337.798049	1337.798049	1337.798049	51.253960	1	1
	3	1337.798049	1337.798049	1337.798049	50.038322	1	1
	4	1337.798049	1337.798049	1337.798049	40.770897	1	1
	5	1337.798049	1337.798049	1337.798049	17.754170	1	1
	6	1337.798049	2316.808305	1439.760609	123.431564	0	0
	7	1337.798049	1337.798049	1337.798049	82.980141	1	1
	8	1337.798049	1337.798049	1337.798049	71.794397	1	1
	9	1337.798049	1337.798049	1337.798049	45.776723	1	1
	10	1337.798049	1337.798049	1337.798049	27.070055	1	1
	Total	13377.980490	14356.990746	13479.943050	601.898634	9	9
	Mean	1337.798049	1435.699075	1347.994305	60.189863		
	STD. Dev	2.39673E-13	309.5902262	32.24339257	32.09368081		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 25	1	1337.798049	1337.798049	1337.798049	56.413583	1	1
	2	1337.798049	1337.798049	1337.798049	137.411909	1	1
	3	1337.798049	1337.798049	1337.798049	93.120650	1	1
	4	1337.798049	1337.798049	1337.798049	27.807537	1	1
	5	1337.798049	1337.798049	1337.798049	101.822336	1	1
	6	1337.798049	2316.808305	1631.501126	154.386858	0	0
	7	1337.798049	1337.798049	1337.798049	48.364147	1	1
	8	1337.798049	1337.798049	1337.798049	39.040267	1	1
	9	1337.798049	1337.798049	1337.798049	57.116447	1	1
	10	1337.798049	1337.798049	1337.798049	33.437622	1	1
	Total	13377.980490	14356.990746	13671.683567	748.921356	9	9
	Mean	1337.798049	1435.699075	1367.168357	74.892136		
	STD. Dev	2.39673E-13	309.5902263	92.87706791	44.55658837		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
PS X 30	1	1337.798049	1337.798049	1337.798049	77.025218	1	1
	2	1337.798049	1337.798049	1337.798049	64.560140	1	1
	3	1337.798049	1337.798049	1337.798049	158.209769	1	1
	4	1337.798049	1337.798049	1337.798049	39.083309	1	1
	5	1337.798049	1337.798049	1337.798049	37.240137	1	1
	6	1337.798049	1337.798049	1337.798049	93.768358	1	1

	7	1337.798049	1337.798049	1337.798049	27.296291	1	1
	8	1337.798049	1337.798049	1337.798049	50.322682	1	1
	9	1337.798049	1337.798049	1337.798049	43.415583	1	1
	10	1337.798049	1337.798049	1337.798049	85.129489	1	1
	Total	13377.980490	13377.980490	13377.980490	676.050976	10	10
	Mean	1337.798049	1337.798049	1337.798049	67.605098		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	38.79142956		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	2545.088972	1672.183927	215.902732	0	0
	2	1337.798049	1337.798049	1337.798049	66.528605	1	1
	3	1337.798049	1337.798049	1337.798049	17.458936	1	1
	4	1337.798049	1337.798049	1337.798049	42.310570	1	1
	5	1337.798049	1337.798049	1337.798049	41.709852	1	1
	6	1337.798049	1337.798049	1337.798049	104.481200	1	1
PS X 35	7	1337.798049	1337.798049	1337.798049	84.573060	1	1
	8	1337.798049	1337.798049	1337.798049	88.969145	1	1
	9	1337.798049	1337.798049	1337.798049	88.328020	1	1
	10	1337.798049	1337.798049	1337.798049	113.758476	1	1
	Total	13377.980490	14585.271413	13712.366368	864.020595	9	9
	Mean	1337.798049	1458.527141	1371.236637	86.402059		
	STD. Dev	2.39673E-13	381.7789115	105.7420991	54.72035859		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	1337.798049	1337.798049	66.007655	1	1
	2	1337.798049	1337.798049	1337.798049	90.985638	1	1
	3	1337.798049	1337.798049	1337.798049	108.630710	1	1
	4	1337.798049	1337.798049	1337.798049	107.786508	1	1
	5	1337.798049	1337.798049	1337.798049	104.118802	1	1
	6	1337.798049	1337.798049	1337.798049	65.524580	1	1
PS X 40	7	1337.798049	1337.798049	1337.798049	210.126398	1	1
	8	1337.798049	1337.798049	1337.798049	42.577924	1	1
	9	1337.798049	1337.798049	1337.798049	51.860789	1	1
	10	1337.798049	1337.798049	1337.798049	69.996940	1	1
	Total	13377.980490	13377.980490	13377.980490	917.615943	10	10
	Mean	1337.798049	1337.798049	1337.798049	91.761594		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	47.77417778		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	1337.798049	1337.798049	63.500767	1	1
	2	1337.798049	1337.798049	1337.798049	60.685961	1	1
	3	1337.798049	1337.798049	1337.798049	91.127541	1	1
	4	1337.798049	1337.798049	1337.798049	104.826251	1	1
	5	1337.798049	1337.798049	1337.798049	68.813601	1	1
	6	1337.798049	1337.798049	1337.798049	54.431423	1	1
PS X 45	7	1337.798049	1337.798049	1337.798049	47.669241	1	1
	8	1337.798049	1337.798049	1337.798049	71.217971	1	1
	9	1337.798049	1337.798049	1337.798049	65.019516	1	1
	10	1337.798049	1337.798049	1337.798049	82.488557	1	1
	Total	13377.980490	13377.980490	13377.980490	709.780830	10	10
	Mean	1337.798049	1337.798049	1337.798049	70.978083		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	17.33131179		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1337.798049	1337.798049	1337.798049	93.525343	1	1
	2	1337.798049	1337.798049	1337.798049	139.171779	1	1
	3	1337.798049	1337.798049	1337.798049	100.601725	1	1
	4	1337.798049	1337.798049	1337.798049	32.347383	1	1
	5	1337.798049	1337.798049	1337.798049	37.905289	1	1
	6	1337.798049	1337.798049	1337.798049	19.862020	1	1
PS X 50	7	1337.798049	1337.798049	1337.798049	252.896888	1	1
	8	1337.798049	1337.798049	1337.798049	114.398341	1	1

	9	1337.798049	1337.798049	1337.798049	47.886038	1	1
	10	1337.798049	1337.798049	1337.798049	26.729047	1	1
	Total	13377.980490	13377.980490	13377.980490	865.323853	10	10
	Mean	1337.798049	1337.798049	1337.798049	86.532385		
	STD. Dev	2.39673E-13	2.39673E-13	2.39673E-13	71.71466519		

Appendix E

Complete Results for 5 Nodes

Table E.1: Complete Results for 5 Nodes

Pop Size 5							
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1323.904429	1323.904429	1323.904429	14.025156	1	0
	2	1317.218365	1317.218365	1317.218365	14.965220	1	1
	3	1317.218365	1317.218365	1317.218365	13.217791	1	1
	4	1317.218365	1317.218365	1317.218365	10.208993	1	1
	5	1317.218365	1317.218365	1317.218365	10.706637	1	1
	6	1317.218365	1317.218365	1317.218365	14.863859	1	1
PS X 5	7	1565.870427	1565.870427	1565.870427	11.513106	1	0
	8	2268.087277	2268.087277	2268.087277	8.780248	1	0
	9	1317.218365	1317.218365	1317.218365	14.414797	1	1
	10	1317.218365	2282.118616	1701.938932	14.843384	0	0
	Total	14378.390690	15343.290941	14763.111257	127.539192	9	6
	Mean	1437.839069	1534.329094	1476.311126	12.753919		
	STD. Dev	301.9438317	398.0069859	309.2875486	2.268014188		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1323.904429	1323.904429	1323.904429	29.982032	1	0
	2	1323.904429	1323.904429	1323.904429	7.582405	1	0
	3	1565.870427	1565.870427	1565.870427	20.109791	1	0
	4	1317.218365	1317.218365	1317.218365	19.441883	1	1
	5	1317.218365	1317.218365	1317.218365	20.668894	1	1
	6	1323.904429	1323.904429	1323.904429	5.837840	1	0
PS X 10	7	1317.218365	1317.218365	1317.218365	6.338859	1	1
	8	2275.920949	2275.920949	2275.920949	2.578905	1	0
	9	1317.218365	1317.218365	1317.218365	24.220025	1	1
	10	1317.218365	1317.218365	1317.218365	14.361994	1	1
	Total	14399.596490	14399.596490	14399.596490	151.122630	10	5
	Mean	1439.959649	1439.959649	1439.959649	15.112263		
	STD. Dev	303.7576474	303.7576474	303.7576474	9.165106881		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Successful
	1	1317.218365	1317.218365	1317.218365	4.443338	1	1
	2	1317.218365	1317.218365	1317.218365	12.481557	1	1
	3	1323.904429	1323.904429	1323.904429	17.419402	1	0
	4	1317.218365	1317.218365	1317.218365	12.471928	1	1
	5	1317.218365	1317.218365	1317.218365	43.748080	1	1
	6	1317.218365	1317.218365	1317.218365	35.439373	1	1
PS X 15	7	1317.218365	1317.218365	1317.218365	41.793918	1	1
	8	1317.218365	1317.218365	1317.218365	15.581427	1	1
	9	1317.218365	1317.218365	1317.218365	27.392444	1	1
	10	1317.218365	1317.218365	1317.218365	25.577376	1	1
	Total	13178.869717	13178.869717	13178.869717	236.348844	10	9
	Mean	1317.886972	1317.886972	1317.886972	23.634884		
	STD. Dev	2.114318978	2.114318978	2.114318978	13.38466695		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	18.093043	1	1

PS X 20

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	2	1317.218365	1317.218365	1317.218365	31.085910	1	1
	3	1323.904429	1323.904429	1323.904429	45.561800	1	0
	4	1323.904429	1323.904429	1323.904429	12.664995	1	0
	5	1317.218365	1317.218365	1317.218365	27.001073	1	1
	6	1317.218365	1317.218365	1317.218365	41.883999	0	0
	7	1317.218365	1317.218365	1317.218365	32.485844	1	1
	8	1317.218365	1317.218365	1317.218365	28.741805	1	1
	9	1317.218365	1317.218365	1317.218365	23.153232	1	1
	10	1317.218365	1317.218365	1317.218365	39.330481	1	1
	Mean	1318.555578	1318.555578	1318.555578	30.000218	9	7
	STD. Dev	2.81909197	2.81909197	2.81909197	10.42192354		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 25	1	1317.218365	1317.218365	1317.218365	12.788970	1	1
	2	1317.218365	1317.218365	1317.218365	34.811245	1	1
	3	1437.525768	1437.525768	1437.525768	6.346825	1	0
	4	1317.218365	1317.218365	1317.218365	18.664745	1	1
	5	1317.218365	1317.218365	1317.218365	24.855022	1	1
	6	1317.218365	1317.218365	1317.218365	19.325382	1	1
	7	1317.218365	1317.218365	1317.218365	19.929891	1	1
	8	1323.904429	1323.904429	1323.904429	63.142352	1	0
	9	1317.218365	1317.218365	1317.218365	10.647082	1	1
	10	1317.218365	1317.218365	1317.218365	60.311510	1	1
	Total	13299.177120	13299.177120	13299.177120	270.823026	10	8
	Mean	1329.917712	1329.917712	1329.917712	27.082303		
	STD. Dev	37.86795851	37.86795851	37.86795851	19.87673865		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 30	1	1317.218365	2281.359613	1510.046615	89.205658	0	0
	2	1317.218365	1317.218365	1317.218365	24.264279	1	1
	3	1323.904429	1323.904429	1323.904429	13.113412	1	1
	4	1317.218365	1317.218365	1317.218365	10.064821	1	1
	5	1317.218365	1317.218365	1317.218365	26.696191	1	1
	6	1317.218365	1317.218365	1317.218365	39.044021	1	1
	7	1317.218365	1317.218365	1317.218365	29.132868	1	1
	8	1317.218365	1317.218365	1317.218365	12.513565	1	1
	9	1317.218365	1317.218365	1317.218365	45.817067	1	1
	10	1317.218365	1317.218365	1317.218365	22.361300	1	1
	Total	13178.869717	14143.010965	13371.697967	312.213181	9	9
	Mean	1317.886972	1414.301096	1337.169797	31.221318		
	STD. Dev	2.114318978	304.6605547	60.77905448	23.3657553		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 35	1	1317.218365	1317.218365	1317.218365	27.757178	1	1
	2	1317.218365	1317.218365	1317.218365	29.207734	1	1
	3	1323.904429	1323.904429	1323.904429	29.186711	1	0
	4	1317.218365	1317.218365	1317.218365	18.650686	1	1
	5	1317.218365	1317.218365	1317.218365	47.020122	1	1
	6	1317.218365	1317.218365	1317.218365	5.116673	1	1
	7	1317.218365	1317.218365	1317.218365	49.087727	1	1
	8	1317.218365	1317.218365	1317.218365	10.691619	1	1
	9	1317.218365	1317.218365	1317.218365	52.114008	1	1
	10	1317.218365	1317.218365	1317.218365	44.062739	1	1
	Total	13178.869717	13178.869717	13178.869717	312.895196	10	9
	Mean	1317.886972	1317.886972	1317.886972	31.289520		
	STD. Dev	2.114318978	2.114318978	2.114318978	16.50355872		

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 40	1	1317.218365	1317.218365	1317.218365	55.723901	1	1
	2	1317.218365	1317.218365	1317.218365	14.459595	1	1
	3	1317.218365	1317.218365	1317.218365	38.024115	1	1
	4	1317.218365	1317.218365	1317.218365	36.733130	1	1

	5	1317.218365	1317.218365	1317.218365	7.656458	1	1
	6	1317.218365	1317.218365	1317.218365	20.646565	1	1
	7	1323.904429	1323.904429	1323.904429	35.465463	1	0
	8	1323.904429	1323.904429	1323.904429	15.069486	1	0
	9	1317.218365	1317.218365	1317.218365	25.008730	1	1
	10	1317.218365	1317.218365	1317.218365	20.013844	1	1
	Total	13185.555781	13185.555781	13185.555781	268.801287	10	8
	Mean	1318.555578	1318.555578	1318.555578	26.880129		
	STD. Dev	2.81909197	2.81909197	2.81909197	14.45205471		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 45	1	1317.218365	1317.218365	1317.218365	19.885583	1	1
	2	1317.218365	1317.218365	1317.218365	62.048248	1	1
	3	1323.904429	1323.904429	1323.904429	16.246160	1	0
	4	1317.218365	1317.218365	1317.218365	44.768325	1	1
	5	1317.218365	1317.218365	1317.218365	11.972276	1	1
	6	1317.218365	1317.218365	1317.218365	15.060771	1	1
	7	1323.904429	1323.904429	1323.904429	4.546297	1	0
	8	1317.218365	1317.218365	1317.218365	16.913525	1	1
	9	1317.218365	1317.218365	1317.218365	68.279116	1	1
	10	1323.904429	1323.904429	1323.904429	35.577228	1	0
	Total	13192.241844	13192.241844	13192.241844	295.297530	10	7
	Mean	1319.224184	1319.224184	1319.224184	29.529753		
	STD. Dev	3.229675586	3.229675586	3.229675586	22.1062441		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 50	1	1317.218365	1317.218365	1317.218365	12.453148	1	1
	2	1317.218365	1317.218365	1317.218365	31.863082	1	1
	3	1317.218365	1317.218365	1317.218365	48.507680	1	1
	4	1317.218365	1317.218365	1317.218365	25.557602	1	1
	5	1317.218365	1317.218365	1317.218365	29.933965	1	1
	6	1317.218365	1317.218365	1317.218365	41.697864	1	1
	7	1317.218365	1317.218365	1317.218365	23.750280	1	1
	8	1317.218365	1317.218365	1317.218365	23.171755	1	1
	9	1317.218365	1317.218365	1317.218365	31.815924	1	1
	10	1323.904429	1323.904429	1323.904429	26.729047	1	0
	Total	13178.869717	13178.869717	13178.869717	295.480346	10	9
	Mean	1317.886972	1317.886972	1317.886972	29.548035		
	STD. Dev	2.114318978	2.114318978	2.114318978	10.04497943		
Pop Size 10							
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 5	1	1317.218365	1317.218365	1317.218365	24.624805	1	1
	2	1317.218365	1317.218365	1317.218365	31.474243	0	0
	3	1317.218365	1317.218365	1317.218365	29.547285	1	1
	4	1317.218365	1317.218365	1317.218365	27.792540	1	1
	5	1317.218365	2268.087277	1412.305257	31.461403	0	0
	6	1317.218365	1317.218365	1317.218365	22.818219	1	1
	7	1317.218365	1420.450768	1327.541606	31.471552	0	0
	8	1317.218365	1317.218365	1317.218365	30.016669	1	1
	9	1317.218365	1317.218365	1317.218365	25.333597	1	1
	10	1317.218365	1317.218365	1317.218365	26.601622	0	0
	Total	13172.183653	14226.284968	13277.593785	281.141934	6	6
	Mean	1317.218365	1422.628497	1327.759378	28.114193		
	STD. Dev	0	298.8302488	29.88302489	3.158498141		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	43.733791	1	1
	2	1323.904429	2281.359613	1515.395466	62.584849	0	0
	3	1317.218365	2275.920949	1413.757230	62.475765	0	0
	4	1317.218365	1317.218365	1317.218365	45.495107	1	1
	5	1317.218365	1317.218365	1317.218365	40.441175	1	1

PS X 10

	6	1317.218365	1317.218365	1317.218365	33.576305	1	1
	7	1317.218365	1317.218365	1317.218365	30.380436	1	1
	8	1317.218365	1317.218365	1317.218365	49.258770	1	1
	9	1317.218365	1317.218365	1317.218365	36.048616	1	1
	10	1317.218365	1317.218365	1317.218365	25.361433	1	1
	Total	13178.869717	15095.027485	13466.899619	429.356245	8	8
	Mean	1317.886972	1509.502749	1346.689962	42.935624		
	STD. Dev	2.114318978	405.3730998	66.59009407	12.57340349		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	83.166170	1	1
	2	1317.218365	1317.218365	1317.218365	91.354764	1	1
	3	1317.218365	1317.218365	1317.218365	59.395657	1	1
	4	1317.218365	1317.218365	1317.218365	45.500610	1	1
	5	1317.218365	1317.218365	1317.218365	29.801186	1	1
	6	1317.218365	1317.218365	1317.218365	60.636053	1	1
PS X 15	7	1317.218365	1317.218365	1317.218365	93.353730	1	1
	8	1317.218365	1317.218365	1317.218365	21.659057	1	1
	9	1317.218365	1317.218365	1317.218365	49.942046	1	1
	10	1317.218365	1317.218365	1317.218365	37.412466	1	1
	Total	13172.183653	13172.183653	13172.183653	572.221739	10	10
	Mean	1317.218365	1317.218365	1317.218365	57.222174		
	STD. Dev	0	0	0	25.29703179		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	55.102413	1	1
	2	1317.218365	1317.218365	1317.218365	63.223244	1	1
	3	1317.218365	1317.218365	1317.218365	51.389844	1	1
	4	1317.218365	1317.218365	1317.218365	56.935671	1	1
	5	1317.218365	1317.218365	1317.218365	56.988651	1	1
	6	1317.218365	1317.218365	1317.218365	56.960002	1	1
PS X 20	7	1317.218365	1317.218365	1317.218365	46.298511	1	1
	8	1317.218365	1317.218365	1317.218365	45.591403	1	1
	9	1317.218365	1317.218365	1317.218365	98.457496	1	1
	10	1317.218365	2301.667285	1609.978408	125.084608	0	0
	Total	13172.183653	14156.632573	13464.943696	656.031844	9	9
	Mean	1317.218365	1415.663257	1346.494370	65.603184		
	STD. Dev	0	311.3100827	92.57885419	25.67440846		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	53.244251	1	1
	2	1317.218365	1317.218365	1317.218365	55.120311	1	1
	3	1317.218365	1317.218365	1317.218365	29.391238	1	1
	4	1317.218365	1317.218365	1317.218365	76.617791	1	1
	5	1317.218365	1317.218365	1317.218365	28.790748	1	1
	6	1317.218365	1317.218365	1317.218365	72.751607	1	1
PS X 25	7	1317.218365	1317.218365	1317.218365	82.252438	1	1
	8	1317.218365	1317.218365	1317.218365	125.657903	1	1
	9	1317.218365	1317.218365	1317.218365	31.948149	1	1
	10	1317.218365	1317.218365	1317.218365	34.434969	1	1
	Total	13172.183653	13172.183653	13172.183653	590.209405	10	10
	Mean	1317.218365	1317.218365	1317.218365	59.020941		
	STD. Dev	0	0	0	31.01279759		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	1	1317.218365	1317.218365	1317.218365	77.924269	1	1
	2	1317.218365	1317.218365	1317.218365	15.673580	1	1
	3	1317.218365	1317.218365	1317.218365	50.870761	1	1
	4	1317.218365	1317.218365	1317.218365	51.571735	1	1
	5	1317.218365	1317.218365	1317.218365	67.300210	1	1
	6	1317.218365	1317.218365	1317.218365	44.682936	1	1
PS X 30	7	1317.218365	1317.218365	1317.218365	158.423012	1	1

Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
	8	1317.218365	1317.218365	1317.218365	100.004655	1	1
	9	1317.218365	1317.218365	1317.218365	144.019693	1	1
	10	1317.218365	1317.218365	1317.218365	27.632366	1	1
	Total	13172.183653	13172.183653	13172.183653	738.103217	10	10
	Mean	1317.218365	1317.218365	1317.218365	73.810322		
	STD. Dev	0	0	0	47.39383397		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 35	1	1317.218365	1317.218365	1317.218365	52.186503	1	1
	2	1317.218365	1317.218365	1317.218365	205.128841	1	1
	3	1317.218365	1317.218365	1317.218365	21.528350	1	1
	4	1317.218365	1317.218365	1317.218365	210.885938	1	1
	5	1317.218365	1317.218365	1317.218365	79.354432	1	1
	6	1317.218365	1317.218365	1317.218365	81.871425	1	1
	7	1317.218365	1317.218365	1317.218365	82.106507	1	1
	8	1317.218365	1317.218365	1317.218365	115.914786	1	1
	9	1317.218365	1317.218365	1317.218365	78.161064	1	1
	10	1317.218365	2275.920949	1604.045773	218.583638	0	0
	Total	13172.183653	14130.886237	13459.011061	1145.721484	9	9
	Mean	1317.218365	1413.088624	1345.901106	114.572148		
	STD. Dev	0	303.1683764	90.70279046	71.13022503		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 40	1	1317.218365	2504.201616	1821.257024	250.832834	0	0
	2	1317.218365	1317.218365	1317.218365	93.543916	1	1
	3	1317.218365	1317.218365	1317.218365	58.917188	1	1
	4	1317.218365	2301.667285	1705.848666	250.466144	0	0
	5	1317.218365	1317.218365	1317.218365	140.320478	1	1
	6	1317.218365	1317.218365	1317.218365	60.813767	1	1
	7	1317.218365	1317.218365	1317.218365	81.046047	1	1
	8	1317.218365	1317.218365	1317.218365	40.098469	1	1
	9	1323.904429	1323.904429	1323.904429	24.908111	1	0
	10	1317.218365	1317.218365	1317.218365	161.776680	1	1
	Total	13178.869717	15350.301888	14071.538676	1162.723633	8	7
	Mean	1317.886972	1535.030189	1407.153868	116.272363		
	STD. Dev	2.114318978	459.9146917	189.8096233	82.35160553		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 45	1	1317.218365	1317.218365	1317.218365	147.284024	1	1
	2	1317.218365	1317.218365	1317.218365	161.154544	1	1
	3	1323.904429	1323.904429	1323.904429	249.658920	1	1
	4	1317.218365	2368.023285	1715.678666	281.548858	0	0
	5	1317.218365	1317.218365	1317.218365	122.226422	1	1
	6	1317.218365	1317.218365	1317.218365	156.792154	1	1
	7	1317.218365	1317.218365	1317.218365	204.180900	1	1
	8	1317.218365	1317.218365	1317.218365	78.763634	1	1
	9	1317.218365	1317.218365	1317.218365	84.424822	1	1
	10	1317.218365	1317.218365	1317.218365	137.984758	1	1
	Total	13178.869717	14229.674637	13577.330018	1624.019036	9	9
	Mean	1317.886972	1422.967464	1357.733002	162.401904		
	STD. Dev	2.114318978	332.0654161	125.7868379	65.91484002		
Max Iterations	Trial	Best Value	Worst Value	Mean	Run Time	Converged	Succesful
PS X 50	1	1317.218365	1317.218365	1317.218365	141.749436	1	1
	2	1317.218365	1317.218365	1317.218365	196.117076	1	1
	3	1317.218365	1317.218365	1317.218365	125.513548	1	1
	4	1317.218365	1317.218365	1317.218365	176.585912	1	1
	5	1317.218365	1317.218365	1317.218365	138.796883	1	1
	6	1317.218365	1317.218365	1317.218365	109.268408	1	1
	7	1317.218365	1317.218365	1317.218365	78.944144	1	1
	8	1317.218365	1317.218365	1317.218365	38.644164	1	1
	9	1317.218365	1317.218365	1317.218365	92.203307	1	1

	10	1323.904429	1323.904429	1323.904429	111.064090	1	0
	Total	13178.869717	13178.869717	13178.869717	1208.886968	10	9
	Mean	1317.886972	1317.886972	1317.886972	120.888697		
	STD. Dev	2.114318978	2.114318978	2.114318978	46.08767708		

Appendix F

Complete Results for 8, 11, and 16 Nodes

Table F.1: Complete Results for 8, 11 and 16 Nodes

# of Nodes	PopSize	Max Itr	Trial No.	Best	Worst	Mean	Run Time	Converged?	Feasible?
			1	3709.364633	3709.364633	3709.364633	2135.775318	1	1
			2	3612.366809	3612.366809	3612.366809	1330.965151	1	1
			3	2768.811719	2768.811719	2768.811719	1744.106020	1	1
			4	3706.083971	3706.083971	3706.083971	1526.288823	1	1
			5	2744.350205	2744.350205	2744.350205	979.239628	1	1
			6	3644.387567	3644.387567	3644.387567	768.772534	1	1
8	100	1000	7	3644.387567	3644.387567	3644.387567	1426.008773	1	1
			8	3726.167111	3726.167111	3726.167111	1473.787969	1	1
			9	3677.343875	3677.343875	3677.343875	2166.887984	1	1
			10	3677.343875	3677.343875	3677.343875	2079.989212	1	1
			Total	34910.607329	34910.607329	34910.607329	15631.821413	10	10
			Mean	3491.060733	3491.060733	3491.060733	1563.182141		
			Std. Dev.	388.6619587	388.6619587	388.6619587	476.831658		

# of Nodes	PopSize	Max Itr	Trial No.	Best	Worst	Mean	Run Time	Converged?	Feasible?
			1	4894.093315	4894.093315	4894.093315	1212.332201	1	0
			2	7484.524339	7484.524339	7484.524339	1213.831208	1	0
			3	6095.846685	6095.846685	6095.846685	1585.760269	1	1
			4	6418.091629	6418.091629	6418.091629	1020.454223	1	0
			5	5466.130557	5466.130557	5466.130557	1678.933603	1	0
			6	6363.405613	6363.405613	6363.405613	1155.457930	1	0
11	100	1000	7	6299.555341	6299.555341	6299.555341	1343.417435	1	0
			8	4450.557589	4450.557589	4450.557589	1981.327153	1	0
			9	4500.255423	4500.255423	4500.255423	2784.428045	1	0
			10	6513.744529	6513.744529	6513.744529	1652.027247	1	0
			Total	58486.205021	58486.205021	58486.205021	15627.969313	10	1
			Mean	5848.620502	5848.620502	5848.620502	1562.796931		
			Std. Dev.	988.9626682	988.9626682	988.9626682	520.9305625		

# of Nodes	PopSize	Max Itr	Trial No.	Best	Worst	Mean	Run Time	Converged?	Feasible?
			1	6236.416191	6236.416191	6236.416191	3175.763427	1	1
			2	4232.927949	4232.927949	4232.927949	4078.338105	1	1
			3	7083.688533	7083.688533	7083.688533	4565.097014	1	1
			4	5270.852907	5270.852907	5270.852907	3998.651640	1	1
			5	6182.653251	6182.653251	6182.653251	5010.905624	1	1
			6	5267.933991	5267.933991	5267.933991	6724.501435	1	1
11	250	25000	7	6211.393347	6211.393347	6211.393347	9951.013579	1	1
			8	5220.283311	5220.283311	5220.283311	3330.842627	1	1
			9	7086.245703	7086.245703	7086.245703	5170.987458	1	1
			10	4259.460147	4259.460147	4259.460147	3325.927473	1	1
			Total	57051.855333	57051.855333	57051.855333	49332.028381	10	10
			Mean	5705.185533	5705.185533	5705.185533	4933.202838		
			Std. Dev.	1024.513635	1024.513635	1024.513635	2066.368842		

# of Nodes	PopSize	Max Itr	Trial No.	Best	Worst	Mean	Run Time	Converged?	Feasible?
			1	8775.008575	8775.008575	8775.008575	1466.738339	1	0
			2	7097.131548	7097.131548	7097.131548	1421.491041	1	0

16	100	1000

			3	8130.996541	8130.996541	8130.996541	1597.926864	1	0
			4	7458.801594	7458.801594	7458.801594	2887.534510	1	0
			5	8594.885009	8594.885009	8594.885009	1528.015340	1	0
			6	7895.464168	7895.464168	7895.464168	1898.791785	1	0
			7	7869.241545	7869.241545	7869.241545	2906.879897	1	0
			8	8171.892469	8171.892469	8171.892469	2262.810357	1	0
			9	8817.957137	8817.957137	8817.957137	1609.730099	1	0
			10	9542.784453	9542.784453	9542.784453	1446.006188	1	0
			Total	82354.163037	82354.163037	82354.163037	19025.924421	10	0
			Mean	8235.416304	8235.416304	8235.416304	1902.592442		
			Std. Dev.	717.4302625	717.4302625	717.4302625	582.8351583		

# of Nodes	PopSize	Max Itr	Trial No.	Best	Worst	Mean	Run Time	Converged?	Feasible?
16	250	2500	1	8946.049503	8946.049503	8775.008575	2367.533035	1	0
			2	7909.209860	7909.209860	7909.209860	4038.915596	1	0
			3	7790.577651	7790.577651	7790.577651	15910.481862	1	0
			4	7777.916541	7777.916541	7777.916541	4835.593004	1	0
			5	8847.405111	8847.405111	8847.405111	11673.866153	1	0
			6	9320.888833	9320.888833	9320.888833	4947.824207	1	1
			7	8255.870347	8255.870347	8255.870347	3797.336509	1	1
			8	7022.167183	7022.167183	7022.167183	2485.055268	1	0
			9	9285.911737	9285.911737	9285.911737	3769.912734	1	1
			10	8702.434843	8702.434843	8702.434843	3166.187469	1	0
			Total	83858.431607	83858.431607	83687.390679	56992.705837	10	3
			Mean	8385.843161	8385.843161	8368.739068	5699.270584		
			Std. Dev.	755.655340	755.655340	743.402688	4462.435278		