B.Sc. in Computer Science and Engineering Thesis

Garbage Collection Routing Based on Traveling Salesman Problem

Submitted by

Md. Mottakin Chowdhury
201205034

Supervised by

Dr. Sadia Sharmin

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

September, 2017

CANDIDATES’ DECLARATION

This is to certify that the work presented in this thesis, titled, “Garbage Collection Routing Based on Traveling Salesman Problem”, is the outcome of the investigation and research carried out by me under the supervision of Dr. Sadia Sharmin.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Md. Mottakin Chowdhury
201205034

CERTIFICATION

This thesis titled, **“Garbage Collection Routing Based on Traveling Salesman Problem”**, submitted by the student as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in September, 2017.

Student Name:

Md. Mottakin Chowdhury

Supervisor:

Dr. Sadia Sharmin
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

I am thankful to Dr. Sadia Sharmin, Assistant Professor of Department of CSE, BUET for her supervision to my thesis. Her ideas and advice regarding my work helped me enormously to make an easier approach to the problem I have studied.

Dhaka
September, 2017

Md. Mottakin Chowdhury

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

Over the years in the field of Computer Science, optimization problems have been very popular due to their connection with real life scenarios. One example of such a scenario where good solution to optimization problems can exploit a huge impact to the whole system is the optimization of the routes for garbage collecting vehicles. This problem has been approached from the perspective of the classical *Traveling Salesman Problem* by many researchers. As a part of the waste management system which is a burning need for providing a healthy lifestyle to the citizens of any country, optimized waste collection can easily contribute to a great deal. Our purpose is to take an attempt to face this problem from a simpler but likely to be efficient perspective. In our approach, we have tried to solve the problem by Dynamic Programming along with accepted heuristics like Double MST and Cristofide's heuristic. We have shown a comparison between them and divided the cases of waste collection into these separate methods. Initially, we have tried to establish the necessity of applying optimization in waste collection from the perspective of Dhaka, the capital city of Bangladesh. In the end, we have proposed a combined method.

Chapter 1

Introduction

As a regular citizen of Dhaka city, sitting in the middle of an ‘infinite’ amount of traffic jam at the busiest hours on a working day, can easily make anyone think about the very common question: *how many people are there in Bangladesh?*

The thing is, this question is very common and of course equally important. We have too many people. And that is even more significantly large comparing the world beyond our border.

When a country has too many people, it has too many things to do. From meeting the basic needs to advanced ones, there are always a very huge amount of things to be done. Since we have a large population, we have large production systems for anything. And this leads us to massive amount of wastes that are created by our people, *every single day.*

So when a country has these many people creating this much amount of wastes throughout the years, it is enormously important to build up a well organized waste management system to get rid of it. That does not necessarily mean that lower population does not need a quality waste management system. It is always important, regardless of the number of population a country may have, to build up a waste management so that the people living in the country can actually lead some comfortable life.

And consequently, one very important task in a waste management system is to collect the solid wastes from garbage bins located around the city and dispose them to a certain place for farther processing. In this regard, we need garbage collection vehicles to do the collection.

Throughout the years of algorithmic research based on real life applications, finding out an optimal *garbage vehicle collection route* has been a popular problem. In our thesis, we take a look at it from different perspectives.

1.1 Waste Collection: Some Basic Ideas

Before jumping into further discussion on our topic, it is in order that we build up some basic ideas regarding waste collection.

According to a report by World Bank [1],

"Waste collection is the collection of solid waste from point of production (residential, industrial commercial, institutional) to the point of treatment or disposal."

This definition points us to one very important direction: we are interested in solid wastes which are, of course, collectible. And these wastes are exclusively denoted as MSW (Municipal Solid Wastes).

1.1.1 Classification of MSW Collection

There are various ways to collect solid wastes in a municipal area. According to the report by World Bank [1], these are some major ways to collect MSW:

- **House-to-House**: Waste collectors visit each individual house to collect garbage. The user generally pays a fee for this service.
- **Community Bins:** Users bring their garbage to community bins that are placed at fixed points in a neighborhood or locality. MSW is picked up by the municipality, or its designate, according to a set schedule.
- **Curbside Pick-Up:** Users leave their garbage directly outside their homes according to a garbage pick-up schedule set with the local authorities (secondary house-to- house collectors not typical).
- **Self Delivered:** Generators deliver the waste directly to disposal sites or transfer stations, or hire third-party operators (or the municipality).
- **Contracted or Delegated Service:** Businesses hire firms (or municipality with municipal facilities) who arrange collection schedules and charges with customers. Municipalities often license private operators and may designate collection areas to encourage collection efficiency.

The above classification of garbage collection methods leads us to another very important direction: we are considering the method of **Community Bins** where municipal corporation sends garbage collection vehicles to collect the wastes disposed by the city-dwellers.

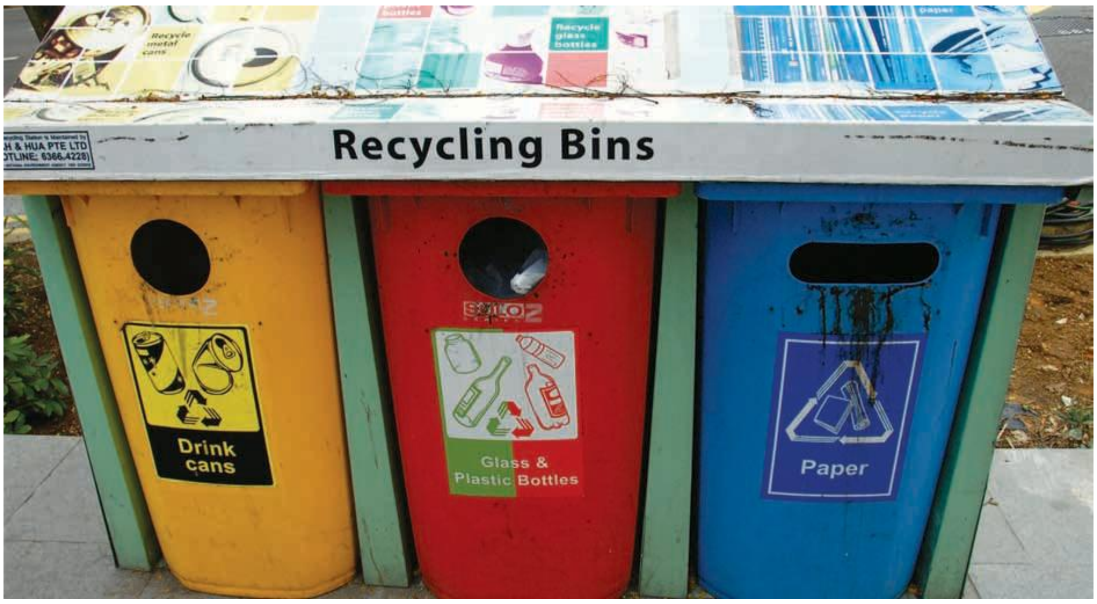

Figure 1.1: *Separate garbage bins in Singapore [1].*

1.1.2 MSW Collection in Developed Cities

Due to the technological advances along with rapid urbanization around the world, it has become incessantly important for the countries to build up an efficient waste management system. To meet this burning need, developed countries have been investing billions of dollars throughout the years. And consequently, the more a country is capable of managing its wastes in an optimal-environment-friendly way, the more it has the scope of providing a healthy life style to its citizens.

As a result, we have seen different developed cities to implement various methods to collect its wastes and dispose them. For example, countries which are more interested in garbage treatment and recycling systems, try to classify the garbage at the very beginning, like in Singapore shown in Figure 1.1.

The garbage collection vehicles follow particular schedules regarding when to come and collect garbage in these cities. There are disposal points where the collected wastes are taken care of by recycling or further processing. The whole system is massive and dynamic. Since the developed countries are investing more and more amount of money in this sector, further improvements in garbage collection is evident.

One of the very popular garbage collection ideas which has been already implemented in many smart cities, is using IoT (Internet of Things) to build up an automated garbage collection system. Many research papers and publications have popularized this IoT based idea over the years and consequently, this is getting much more attention. The idea is quite interesting and realistic. According to [6],

”The system consists of smart containers or smart bins, each bin or container installed with

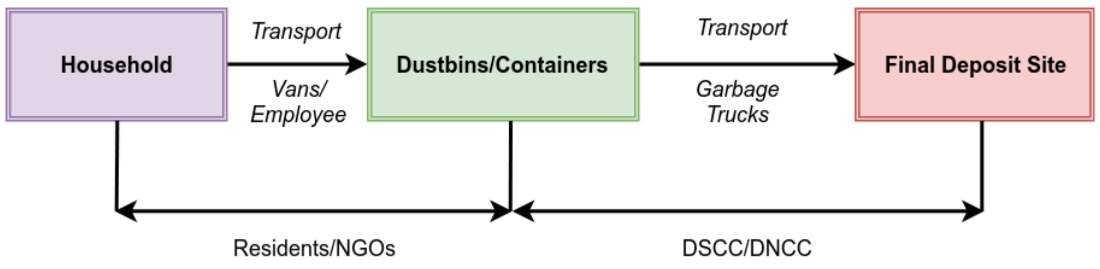

Figure 1.2: *Garbage collection in Dhaka city [2].*

Arduino Uno, ultrasonic sensor and Radio Frequency (RF) transmitter on the top of the container. When the container is full of waste, it sends signal to the control center which will have the level of waste in the containers and through GSM/GPRS , a message (SMS) will send to the mobile phone of the truck driver of which waste bin is full and need to be emptied."

So what is the motivation behind using a full-fledged IoT system in garbage bins? It is actually quite intuitive. When garbage bins get overflown, the environment around them starts to get affected. As a result, it is really important to collect the garbage from these bins as soon as they are full. This job is done by using IoT - the buzzword in today's world of technological wonders.

1.1.3 MSW Collection in Dhaka

Unfortunately, we have a very different and unexpected scenario in the capital city of Bangladesh. The MSW collection system is not very well organized in Dhaka which leads us to suffer from serious environmental crises regarding solid waste management. In spite of being the heart of the country, the waste management and collection system has not been developed as much as the other aspects of the city.

According to [7], Dhaka city is now producing 4600-5110 tons of waste per day. There are two administrative units, Dhaka North City Corporation (DNCC) and Dhaka South City Corporation (DSCC), who are in charge of collecting waste and its disposal in their particular ranges. But very unfortunately, only approximately 50 percent of the total wastes produced are collected and processed by these two administrative units.

In this paper [7], the methodology of MSW collection in Dhaka is categorized and then analyzed based on environmental perspective. Their study area was the 33 no. ward of DSCC.

The methods discussed in the paper are noted below:

- **Household Waste Collection by Vans:** In this method, several vans (like in Figure 1.3 are sent to the households of a locality to collect the garbage produced in each of the houses.

Figure 1.3: *Waste collecting vans [3].*

- **Household Waste Disposal by Employment:** In many regions, people appoint someone to collect the wastes from the households to dispose them to the nearest garbage disposal bins established by the authority.
- **Roadside Dumping Areas:** There are some roadside dumping areas around the city where dwellers dump their wastes. Unfortunately, these roadside dumping areas are extremely unhygienic and injurious to the environment.
- **Roadside Garbage Bins:** Authority of a locality establishes garbage bins in a suitable place where dwellers can easily dump their wastes.
- **Garbage Collection Vehicles:** Following different schedules, garbage collection vehicles visit all these dumping places to collect the MSW produced. They do not follow very definite schedule. And more surprisingly, it is very common that no garbage collection vehicles come to collect wastes for a long time.

So overall, the garbage collection procedure that we are interested in can be depicted in Figure 1.2 based on [2]. In this figure, we see two particular sections into which the whole collection system is divided into. One part is handled by the residents or some NGOs where the other part is the concern of two city corporations of Dhaka.

Our purpose is to find out optimal routes for the garbage trucks maintained by DNCC and DSCC. The next section briefly tries to describe the motivation behind our work. Further chapters approach to our problem gradually.

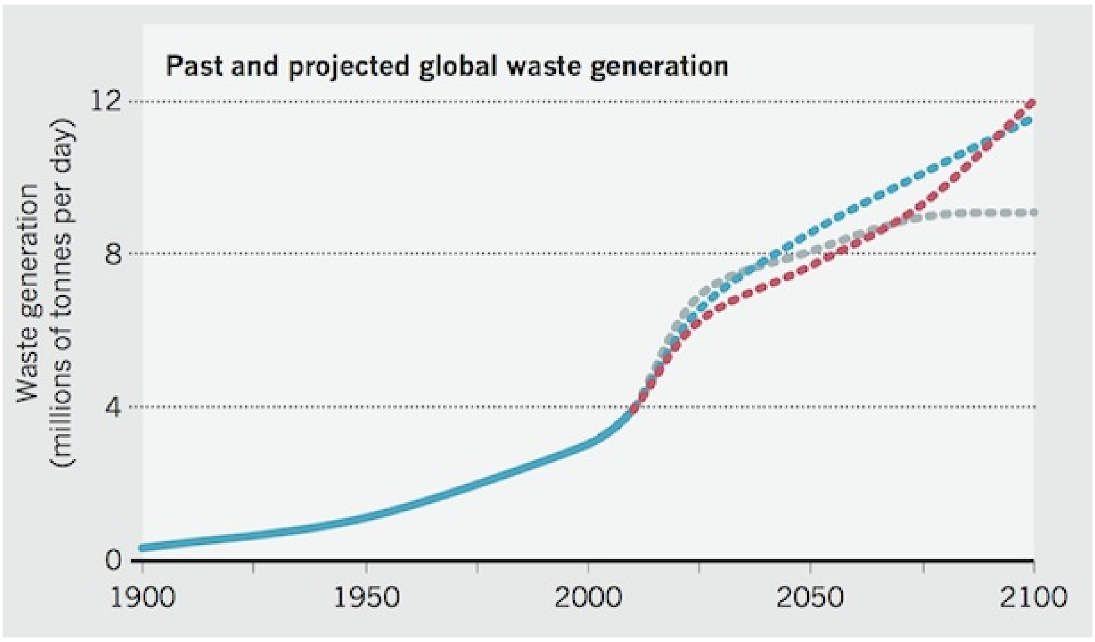

Figure 1.4: *Global waste generation*

1.2 Motivation

It is very evident that efficient waste management is a burning need. In this section, we try to build up the motivation regarding the necessity of optmizing the grabage collection routing.

1.2.1 Waste Generation: Where is the World Heading?

According to [1], current global MSW generation levels are approximately 1.3 billion tonnes per year, and are expected to increase to approximately 2.2 billion tonnes per year by 2025. This represents a significant increase in per capita waste generation rates, from 1.2 to 1.42 kg per person per day in the next fifteen years. However, global averages are broad estimates only as rates vary considerably by region, country, city, and even within cities.

There are practically two scenarios in current world regarding waste management. In one part of the world, some countries are showing concern in developing a full functioning waste management system. Still some of these countries like China or USA are having a hard time to achieve their purpose. On the other hand, there are some other countries which are less concerned regarding the waste management crisis they are facing or they will have to face in near future.

The only one word that can actually describe the scenario of the world is: *alarming*. This earth is a *pale blue dot* [8] of seven billion people. The amount of wastes that this amount of people are responsible for producing every day is simply staggering.

We need a way to manage the wastes we are responsible for. Not for the sake of the environment,

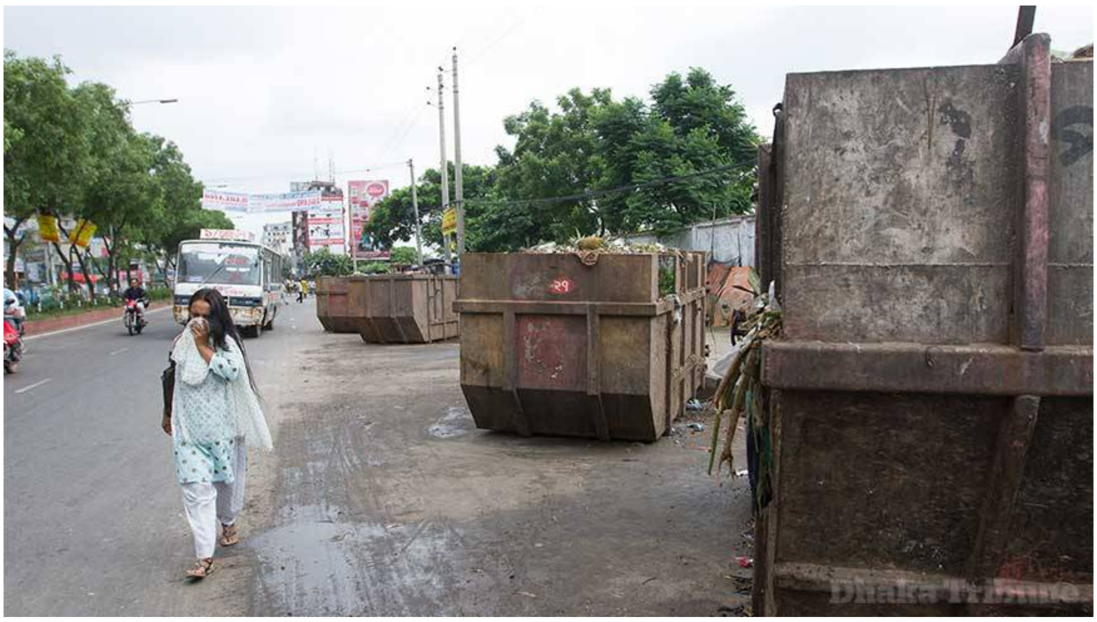

Figure 1.5: *Garbage bins creating public inconvenience [4].*

but for the sake of humanity.

And in this regard, a small improvement in MSW collection can contribute to the whole management system in a great deal. From the perspective of the whole world, this is surely evident.

1.2.2 A Brief Scenario of MSW Management in Dhaka

The idea which we tried to build up in the previous section is that the city of Dhaka has a huge population with even huger MSW production every day. Solid waste management, without any doubt, represents a very concerning issue due to the fact that it causes land contamination if transparently dumped, water contamination if dumped in the swamps and air contamination if smoldered or burnt openly. Dhaka city is confronting serious environmental imbalance because of the uncollected transfer of waste on avenues and other open territories, obstructed seepage by tainting of water assets close uncontrolled dumping locales. Almost all the area of this city does not experience adequate waste management [7].

On top of that, approximately about 40 percent waste is collected by the authority (DNSC and DSCC), but uncollected waste is dropped in open dustbins beside the roads which is of course, very disastrous to the environment. Considering the Dhaka city's fast development and lacking waste management, the requirement for enhanced strong waste management shows a key open door for concurrently tending to the environmental and health issue [9]. There is a variation of waste generation between Old Dhaka and New Dhaka. Dhaka creates roughly 1.65 million metric huge amounts of strong waste every year. Per person waste generation is somewhere around 0.29 and 0.60 kilograms dependent upon the people of various income levels [10].

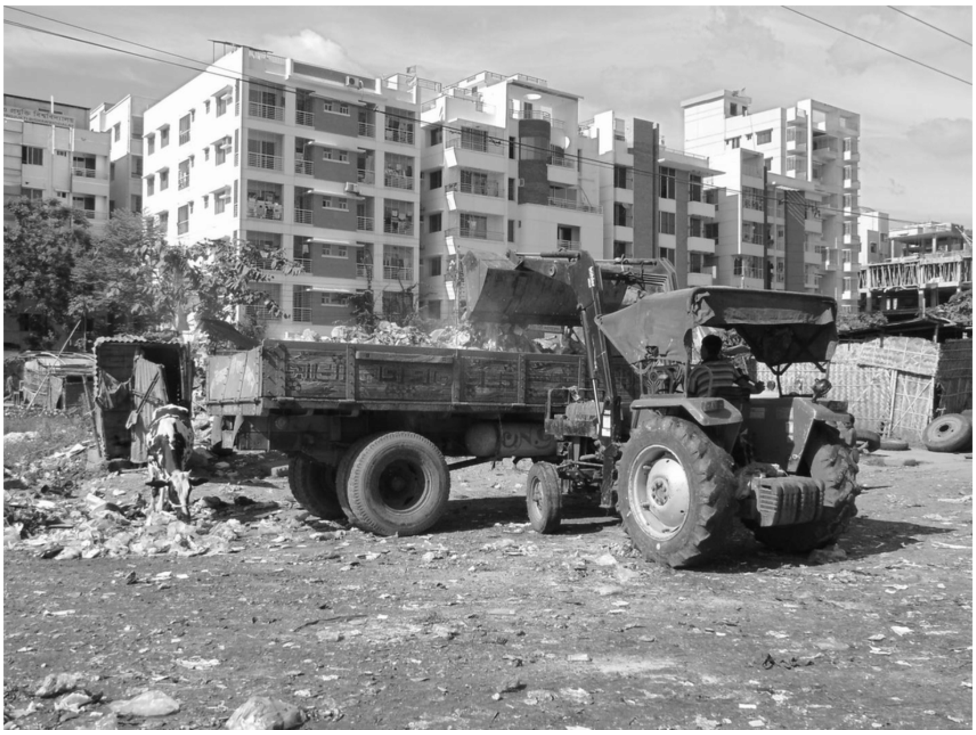

Figure 1.6: *Trucks disposing of garbage in a location near Uttara, Dhaka.*

On the other hand, it is not very easy to expect that the authorized body is performing the duty of collection in a well-organized way. Apart from schedule failure and overflowing bins, there is the problem of inconsiderate allocation of garbage bins. As mentioned in the report [4], the authority is failing to introduce proper locations for the garbage bins to be established. This failure in management is responsible for public nuisance every day. Then again, people of some localities themselves can be responsible for the unhealthy environment they are living in. Specially it is very common to find out people throwing their garbage to some random corner which eventually turn into a garbage dumping area.

So it should be clear by now that to manage this tremendous amount of waste, we need a fairly efficient waste management as well as collection system. The more we can optimize the collection procedure, the more it will be easier for us to manage the wastes through further treatment and processing. There are garbage collecting trucks which are deployed by the central authority (DNSC and DSCC) to do the collection but there are scopes to improve the whole procedure. We are interested in optimizing the route for these vehicles in our thesis.

It should be noted that to solve all those problems mentioned till now is not very easy a task. There are lots of factors creating significant amount of embankments in the process of solving all these problems. Starting from the amount of population and huge waste production the authority needs to deal with to the unhelpful behaviour of city dwellers - lots of such scenarios are responsible for the mismanagement we face all the time.

1.2.3 The Last Piece of Motivation

So finally, what we want to establish is that waste management in an efficient way is a burning need in the perspective of current worlds environmental and infrastructural scenario. In developed cities around the world, this issue is regarded as the few of the most important issues and thousands of projects as well as researches are being conducted in a massive scale. As an emerging nation of the future world, we believe that Bangladesh should look for methodical ways to collect and manage its wastes. Not only because of the current needs we are dealing with, but also because of the futurist necessities we will encounter. As a part of meeting the upcoming challenges Bangladesh will have to deal with in near future, we are interested in improving garbage collection methods based on algorithmic approach.

In the next chapter we try to build up some technical ideas or backgrounds which are exclusively important for approaching towards a solution for the problem we are interested in. We will also have a look at various approaches done by other researchers as well as the algorithms developed by them in further chapters.

Chapter 2

Background

In the previous chapter, we lighted upon the garbage collection basic concepts along with the current situation of Dhaka city. To build up a better waste management system for our city, we need a lot of technological implementations in this sector.

It has already been noted several times that we are in quest of finding out an optimal route for the garbage collection trucks that are deployed in the city. In order to complete our quest for solution, it is in order to discuss the technical ideas. In this chapter, we aim at building the necessary technical background to further proceed towards the solution of our problem.

2.1 Notations

Some frequently used notations in this paper are noted below:

- V - a set of nodes/vertices.
- E - a set of edges built up using V.
- $G(V, E)$ - denotes a graph of V and E. If it is not mentioned explicitly, $G(V, E)$ is always an un-directed graph.
- $C(u, v)$ - denotes the cost between two vertices u and v. If not mentioned otherwise, $C(u, v) = C(v, u)$ which is intuitive in case of un-directed graphs.
- $MST(G)$ - the Minimum Spanning Tree of $G(V, E)$.

Less frequently used notations will be introduced in the respective sections.

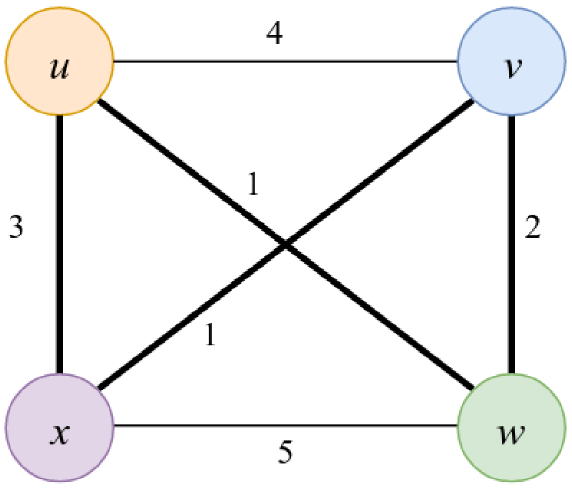

Figure 2.1: *An instance of TSP. The thick lines denote the minimum cost tour, with cost 7 [5].*

2.2 The Traveling Salesman Problem (TSP)

One of the very popular and universally studied problems of Computer Sciences would be Traveling Salesman Problem (TSP). This is the core foundation of our thesis work.

2.2.1 Definition

TSP has been defined in [5] as:

In the traveling-salesman problem, which is closely related to the Hamiltonian-cycle problem, a salesman must visit n cities. Modeling the problem as a complete graph with n vertices, we can say that the salesman wishes to make a tour, or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. The salesman incurs a non-negative integer cost $c(i, j)$ to travel from city i to city j, and the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour.

As an example, we can check out Figure 2.1. The minimum-cost tour in this example is (u, w, v, x, u) with cost 7.

In the definition noted above, the graph G is modeled as a complete graph. The cost function c is a function from $V \times V \to \mathbb{Z}$. And if the G has a traveling-salesman tour with cost at most $k \in \mathbb{Z}$, then

$TSP = (G, c, k)$.

2.2.2 NP-completeness of TSP

TSP is **NP-complete**. This is a kind of bad news as this class of problems are not easy to solve. We cannot solve this problem quickly. To be more formal, we have no polynomial time solution to solve this problem. As a result, over the years we have approached this problem using [11]:

- **Heuristics:** If we cannot quickly solve the problem with a good worst case time, we can at least try to use some heuristics to solve a reasonable fraction of the common cases.
- **Approximation:** A lot of the time it is possible to come up with a provably fast algorithm, that doesn't solve the problem exactly but comes up with a solution you can prove is close to right.
- **Exponential Time Solution:** Trying to solve the problem with an exponential time solution is also used in mostly small cases.

The proof for the NP-completeness of TSP can be found in [5] provided in **Theorem 34.14**. The proof is conducted by first showing that TSP is NP, then it is proved to be an NP-hard based on its connection with Hamiltonian cycle.

To be noted that the naive solution complexity of TSP is $\mathcal{O}(n!)$. In this naive approach, all different permutation of the nodes are considered and the tour cost following each permutation is calculated. Then we take the minimum one.

Not to mention that very rarely this naive approach is used. A better approach is using *Dynamic Programming* with *bitmasks*. This approach is further discussed later.

2.3 Classification of TSP

TSP can be divided into three classes based on the salesmen or nature of the graph. They are [12]:

- **sTSP:** Symmetric TSP. Let $V = v_1, v_2, ..., v_n$ be a set of cities, $E = \{(r, s) : r, s \in V\}$ be the edge set, and $d_{rs} = d_{sr}$ be a cost measure associated with $(r, s) \in A$. The $sTSP$ is the problem of finding a minimal length closed tour that visits each city once. If the cities $v_i \in V$ are given by their coordinates (x_i, y_i) and d_{rs} is the *Euclidean distance* between r and s, then we have something called *Euclidean* TSP.
- **aTSP:** Asymmetric TSP. This is same as $sTSP$ but only opposite in one criteria. At least for one edge (r, s), $d_{rs} \neq d_{rs}$.

- **mTSP:** Multiple TSP. In a given set of nodes, let there are m salesmen located at a single depot node. The remaining nodes (cities) that are to be visited are intermediate nodes. Then, the $mTSP$ consists of finding tours for all m salesmen, who all start and end at the depot, such that each intermediate node is visited exactly once and the total cost of visiting all nodes is minimized. The cost metric can be defined in terms of distance, time, etc.

  There are some variations of $mTSP$ which can be of some interest in our thesis:

  – *Single vs. Multiple Depots:* In the single depot, all salesmen finish their tours at a single point while in multiple depots the salesmen can either return to their initial depot or can return to any depot keeping the initial number of salesmen at each depot the same after the travel is finished.

  – *Number of Salesmen:* The number of salesman in the problem can be fixed or a bounded variable.

  – *Cost:* When the number of salesmen is not fixed, then each salesman usually has an associated fixed cost incurring whenever this salesman is used. In this case, the minimizing the requirements of salesman also becomes an objective.

  – *Timeframe:* Here, some nodes are needed to be visited in a particular time period that is called time window which is an extension of the $mTSP$, and referred as *Multiple Traveling Salesman Problem* with specified *timeframe* $(mTSPTW)$. We can easily relate the research on garbage collection within time-window with $mTSPTW$.

  Interestingly, different problems regarding vehicle routing are based on $mTSP$. We have seen approaches to the garbage collection routing problem based on $mTSP$ along with fixed time-windows.

2.4 Linkage of TSP with Real Life Applications:

It is obvious that TSP will have real-life applications. Below we discuss couple of them:

- **Drilling of Printed Circuit Boards:** A direct application of the TSP is in the drilling problem of printed circuit boards (PCBs). To connect a conductor on one layer with a conductor on another layer, or to position the pins of integrated circuits, holes have to be drilled through the board. The holes may be of different sizes. To drill two holes of different diameters consecutively, the head of the machine has to move to a tool box and change the drilling equipment. This is quite time consuming. Thus it is clear that one has to choose some diameter, drill all holes of the same diameter, change the drill, drill the holes of the next diameter, etc. Thus, this drilling problem can be viewed as a series of

Figure 2.2: *PCB drilling.*

TSPs, one for each hole diameter, where the *cities* are the initial positions and the set of all holes that can be drilled with one and the same drill. The *distance* between two cities is given by the time it takes to move the drilling head from one position to the other. The aim is to minimize the travel time for the machine head. [13].

- **Overhauling Gas Turbine Engines:** It occurs when gas turbine engines of aircraft have to be overhauled. To guarantee a uniform gas flow through the turbines there are nozzle-guide vane assemblies located at each turbine stage. Such an assembly basically consists of a number of nozzle guide vanes affixed about its circumference. All these vanes have individual characteristics and the correct placement of the vanes can result in substantial benefits (reducing vibration, increasing uniformity of flow, reducing fuel consumption). The problem of placing the vanes in the best possible way can be modeled as a TSP with a special objective function [14].

- **X-Ray Crystallography:** Analysis of the structure of crystals [15] is an important application of the TSP. Here an X-ray diffractometer is used to obtain information about the structure of crystalline material. To this end a detector measures the intensity of X- ray reflections of the crystal from various positions. Whereas the measurement itself can be accomplished quite fast, there is a considerable overhead in positioning time since up to hundreds of thousands positions have to be realized for some experiments. In the two examples that we refer to, the positioning involves moving four motors. The time needed to move from one position to the other can be computed very accurately. The result of the experiment does not depend on the sequence in which the measurements at the various positions are taken. However, the total time needed for the experiment depends on the sequence. Therefore, the problem consists of finding a sequence that minimizes the total positioning time. This leads to a TSP.

- **Computer Wiring:** [16] reported a special case of connecting components on a computer board. Modules are located on a computer board and a given subset of pins has to be connected. In contrast to the usual case where a Steiner tree connection is desired, here the requirement is that no more than two wires are attached to each pin. Hence we have the problem of finding a shortest Hamiltonian path with unspecified starting and terminating points. A similar situation occurs for the so-called testbus wiring. To test the manufactured board one has to realize a connection which enters the board at some specified point, runs through all the modules, and terminates at some specified point. For each module we also have a specified entering and leaving point for this test wiring. This problem also amounts to solving a Hamiltonian path problem with the difference that the distances are not symmetric and that starting and terminating point are specified.

- **The Order-picking Problem in Warehouses:** This problem is associated with material handling in a warehouse [17]. Assume that at a warehouse an order arrives for a certain subset of the items stored in the warehouse. Some vehicle has to collect all items of this order to ship them to the customer. The relation to the TSP is immediately seen. The storage locations of the items correspond to the nodes of the graph. The distance between two nodes is given by the time needed to move the vehicle from one location to the other. The problem of finding a shortest route for the vehicle with minimum pickup time can now be solved as a TSP.

- **Vehicle Routing:** Suppose that in a city n mail boxes have to be emptied every day within a certain period of time, say one hour. The problem is to find the minimum number of trucks to do this and the shortest time to do the collections using this number of trucks. As another example, suppose that n customers require certain amounts of some commodities and a supplier has to satisfy all demands with a fleet of trucks. The problem is to find an assignment of customers to the trucks and a delivery schedule for each truck so that the capacity of each truck is not exceeded and the total travel distance is minimized. Several variations of these two problems, where time and capacity constraints are combined, are common in many real- world applications. This problem is solvable as a TSP if there are no time and capacity constraints and if the number of trucks is fixed (say m). In this case we obtain an $mTSP$ [16].

The last application of TSP interests us the most. It has long been studied as a solution approach for various kinds of Vehicle Routing Problems (VRPs).

The sections written above were intended to building up concepts regarding TSP and its real-life applications. The next section gives us a *literature review* on our topic.

2.5 Literature Review

In this section, we intend to give a look back to the previous work related to our topic.

In [18], authors solve the waste collection routing problem from individual inhabitants. This type of waste collection routing problems is determined as kerbside collection. This problem was modified to the problem of the assignment of vehicles to tasks. The presented method in this paper designates the set of tasks which are assigned to each vehicle, so it designates the routes of vehicles. The method consists of two stages. In the first stage the tasks were designated, whereas in the second stage the assignment of vehicles to tasks was made.

In order to solve the assignment problem, the authors considered several variables like: *set of nodes, size of waste, waiting time in loading and unloading, distance between the points in the transportation network, number of drivers, number of trucks, expected unloading time etc.*

Then they tried to optimize the cost based on the following optimization function:

$F(X,Y) = KZZ(X) + KZP(X) + KPZ(Y) + KPP(Y)$

Where, $KZZ(X)$ - the costs related to the fuel consumption in the tasks, $KZP(X)$ - the costs related to the hourly rate of a driver, $KPZ(Y)$ - the costs related to the fuel consumption in the assignment route, $KPP(Y)$ - the costs related to the hourly rate of a driver in the assignment route.

Finally, to do the optimization phase, the authors used a *hybrid algorithm* as a heuristic.

On the other hand, the paper [19] proposed a waste collection system based on positions of waste bins, the road network and the population density in the area under study. In addition, waste collection schedules, truck capacities and their characteristics were also taken into consideration. The authors basically used MST to locate the shortest possible routes to collect waste from all dustbins in their study region. They took a small part of Kolhapur city as the case study area. The Corporation of Kolhapur has divided its area in about 11 sanitary wards solid waste collecting programs, where in general each one includes approximately 100 waste bins. Any garbage truck that is responsible for the collection of the solid waste in that given area must visit all the bins in order to complete its collection program.

After collecting data and building up a garbage bins map of the study region, the MST of the study region was built up. This paper proposed the route of MST to be an optimized route for their study area.

Genetic Algorithm was used in the paper [20]. In this paper, the proposed MSW management system is based on a geo-referenced spatial database supported by a geographic information system (GIS). The GIS takes into account all the required parameters for solid waste collection. These parameters include static and dynamic data, such as the positions of waste bins, the road

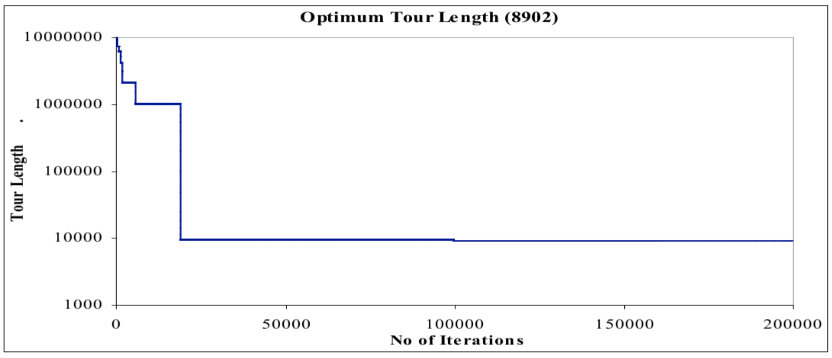

Figure 2.3: *The best result in GA Algorithm.*

network and its related traffic, as well as the population density in the area under study. In addition, waste collection schedules, truck capacities and their characteristics were also taken into consideration in their work. Spatio-temporal statistical analysis was used to estimate inter-relations between dynamic factors, like network traffic changes in residential and commercial areas. The user, in the proposed system, was able to define or modify all of the required dynamic factors for the creation of alternative initial scenarios. The objective of the system was to identify the most cost-effective scenario for waste collection, to estimate its running cost and to simulate its application.

Like the previous research paper, the authors also chose a small part of Atticas prefecture (a suburb of Athens) as the case study area.

In order to execute the GA proposed by the authors in [20], they adjusted the values of the following six parameters:

- *Iterations*
- *Population*
- *Children per Generation*
- *Mutation Policy*
- *Mutation Probability*
- *Diversity Threshold*

They ran simulations on the data based on their study area and got the best result for following values:

- *Iterations:* 200000
- *Population:* 100

- *Children per Generation:* 5
- *Mutation Policy:*
  - Swap 2 Loading Spots - the 1st 100000 iterations (9084) and
  - Reverse Subtour - the 2nd 100000 iterations (8902).
- *Mutations Probability:* 0.1
- *Diversity Threshold:* 0.01

The best result of their GA approach is shown in Figure 2.3.

Genetic Algorithm approach was also used in [21]. In this research paper, the authors tried to find the best route for collecting solid waste in cities taking *Irbid City* in Jordan as an example problem. The routing problem in this example was a node routing or of course, a TSP. They tried to develop the method based on a real Genetic Algorithm.

The authors claim that the results of their study, in comparison with the other applied optimization methods (linear, dynamic, Monte Carlo and heuristic search method), indicate that the real GA through its specific behavior and through its efficient operators, produces significantly the lowest distance (cost tour) solution. They also concluded that real GA approach is robust, represents an efficient search method and is easily applied to dynamic and complex system of the well-known TSP in the field of solid waste routing system in the large cities.

We have seen an interesting approach to face the problem of solid waste collection in [22]. This paper deals with the solid waste collection efficiency improvement by solving optimized route using TSP and *Particle Swarm Optimization* (PSO) algorithm. According to the authors of this paper, *Vehicle Routing Problem* (VRP) for solid waste collection using PSO is a new concept. Apart from using PSO, route optimization is also modeled considering different scenarios and constraints such as *time window, vehicles maximum capacity, percentage of waste level,* etc. to find the most efficient route to collect the solid waste.

To be noted that *Particle Swarm Optimization* is an evolutionary optimization algorithm which is basically a population based optimization method simulating social behavior of flocks of birds searching for foods. This method of optimization for difficult problems was proposed in 1995 by *Kennedy and Eberhart* [23].

In this particular paper, the authors claim that their proposed method finds out the best path for solid waste collection using PSO algorithm applying the concept of TSP. They used an optimization function for their approach:

$Z_l = \min \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=1}^{v} C_{ij} * x_{ij}^{k}$

where,

C_{ij} - cost of the edge between bins i and j. v - number of vehicles.

$$x_{ij}^{k} = \begin{cases} 1, \textit{if vehicle k travels directly from i to j} \\ 0, \textit{otherwise} \end{cases}$$

The author of this paper finally show a good optimized route for collecting waste by applying PSO. They did the experiment for 25 waste bins. They claimed that the distance obtained by applying PSO is reasonable.

All of the papers mentioned above tried to tackle the routing problem based on TSP. But in [24], the problem is viewed from the perspective of *Chinese Postman Problem (CPP)* where the garbage trucks are needed to visit edges corresponding to the garbage bins. They finally used *Artificial Immune System* as a heuristic to get a solution for the problem.

Simulation based approaches have also been taken to solve this problem like in [25]. This paper presented a real-world case study of optimizing waste collection in Sweden. The problem, involving approximately 17,000 garbage bins served by three bin lorries, was approached as a TSP and solved using simulation-based optimization and an *Evolutionary Algorithm*.

And also in [26], *Evolutionary Algorithm* was used to solve the problem of vehicle routing.

Lastly, another popular form of optimization technique to solve the TSP for vehicle routing is *Ant Colony Optimization (ACO)* which is used in many papers, like in [27].

In the next chapter, we finally try to light upon the attempts taken by us to solve the problem.

Chapter 3

Our Approach

We have already discussed in previous chapters that TSP is our main concern in case of solving the problem we are interested in. Since no polynomial time solution is possible for solving any form of TSP, *heuristics* or *approximation* approaches had always been the best thing researchers could do. Not to forget that there are differences and variations among the *heuristics* from the perspective of *performance*, *computation time* and *implementation difficulty*.

This chapter will try to discuss the efforts taken by us to deal with the problem of garbage vehicle routing for municipal solid waste collection.

3.1 Graph Representation

The representation of the map of the garbage bins as a graph is the first important thing to actually solve the problem. After considering couple of options, we finally decided on the following representation:

- Each garbage bin is a node of the graph.
- The edges are the roads connecting the garbage bins. One key idea is that these edges are bidirectional. It is because if a garbage truck visits from bin 1 to bin 2 and then comes back to 2 to 1, it is very likely that the cost in both direction will be the same.
- The bins can be uniquely denoted by their satellite coordinates and then numbered as $1, 2, 3..., n$. This numbering can be helpful to implement the solution but we can still uniquely identify the bins from their underlying entity (satellite coordinates).
- The cost of an edge is the cost of a garbage truck to visit from one bin to another following that particular edge. This cost can be *fuel consumption*, *time needed*, *distance of the bins* etc. Further discussion on the cost of an edge will follow.

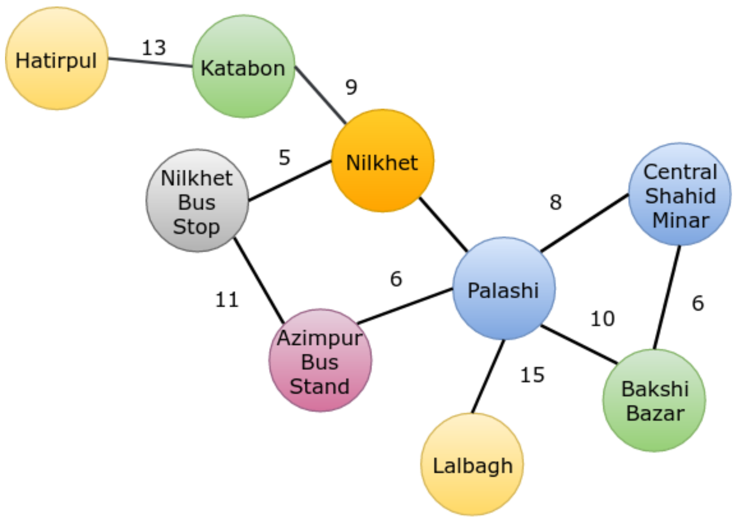

Figure 3.1: *Graph representation of the garbage bin map.*

It is to be noted that we can actually denote a bin with some physical address attributed to each of them. For example, a garbage bin situated on the side of the road connecting *Nilkhet* and *Palashi Circle* in front of *University of Dhaka Residential Quarter* can be denoted as *the bin in front of DU Quarter* etc.

Important thing is that the representation of the nodes is not much of a problem. There are various ways possible for us to represent the nodes and we can choose any of them as we need.

The Figure 3.1 shows a graph representing the map of garbage bins in Dhaka. This is not based on an actual garbage bin map of the city rather based on *Google Maps*. In this figure, the numbers written alongside the edges denote the cost of traveling between the nodes. Of course, the costs are randomly chosen.

Below is an even simpler representation of garbage bin network as a graph. Here all the nodes, edges and costs are randomly chosen.

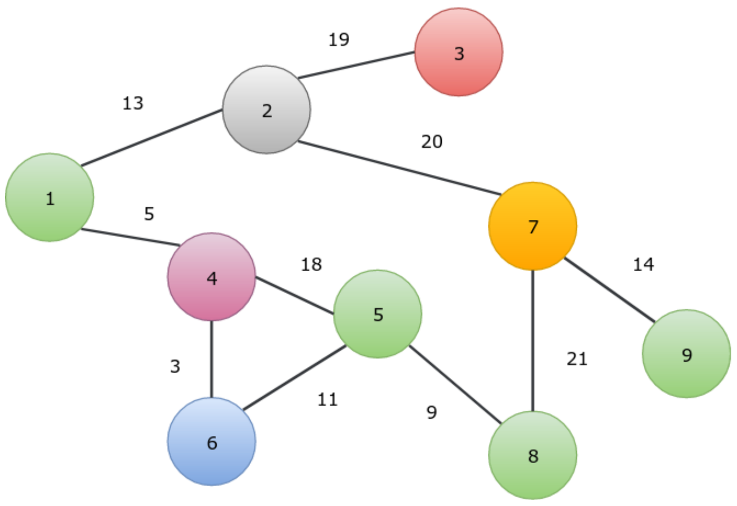

Figure 3.2: *Simpler representation of the garbage bin network.*

3.2 Input Procedure

The input of the problem will be of the form below:

N - number of nodes (garbage bins) in our network M - number of edges (paths/roads) connecting the nodes

Then goes M sets of data each of the form:

u v w - where u is the *starting node*, v is the *ending node* and w is the cost of visiting from u to v.

For example, below is sample input:

Table 3.1: Input graph for Figure 3.2.

		Edges		
N	M	u	v	w
9	10	1	2	13
		2	3	19
		2	7	20
		7	9	14
		7	8	21
		5	8	9
		5	6	11
		5	4	18
		4	6	3
		1	4	5

3.3 Approach Using Dynamic Programming

One very much popular way to solve problems regarding *exhaustive search* is **Dynamic Programming** or **DP** in short. In this section, we describe our attempt to solve the corresponding problem of vehicle routing using DP.

Suppose we are considering a particular garbage collecting vehicle for which we are interested in finding out an optimal route to do its job. Let us assume that a particular number of garbage bins are assigned to the vehicle and the particular locations of the garbage bins are known to the driver of the vehicle.

To collect the garbage located in the bins, the truck needs to visit each of the bins one at a time. This means that the driver has to visit all of the nodes of the graph. It can be thought of this way: each of the vehicle has its own graph to which the vehicle is assigned.

So, we have a graph consisted of *nodes, edges* and *costs* corresponding to the edges. For the sake of clarity, we assume that there is a particular node from where the truck starts its journey, it visits all the nodes in its corresponding graph and then comes back to that particular node. It can be the case that there is a starting node or *parking place* from where the truck starts its journey and after collecting its assigned task, the truck has to go to some particular ending node or *depot* where it gets rid of its collected wastes. Separate depot and parking spot will not be a problem as we can just follow the shortest path from depot to the parking place after disposal.

Now, we assume that in our graph $G(V, E)$ we have n nodes, m edges and a cost function $c(u, v)$ from nodes u to node v. Below is an explanation for the DP approach we have implemented. Pseudocode is provided in Algorithm 1.

The technique we are going to use is known as *DP with bitmasks*.

In the technique of *bitmask DP*, we use a combination of bits which is called *bitmask* or more conveniently $mask$ to represent the combination of nodes we have visited till now. This $mask$ will be one of the states for our DP approach. The second state that we use is *last visited node* or $last$. This is simply what the name suggests, the index or id of the last node we visited. From this $last$ node, we are interested in traveling to the any of the nodes which is unvisited.

In Algorithm 1, we can see two functions. In the *main* function, we take input and call the *recursive* function $PROCDP$. Before calling this procedure, we take two variables $mask$ and $start\_node$. $mask$ denotes which combination of nodes has already been visited and $start\_node$ denotes the starting place of the garbage truck from where it starts its journey. Here, $mask$ equals 1 because the truck has visited the 0 node or the $starting\_node$ already.

Now we call the *recursive* function $FUNCTIONDP$ setting $mask$ and $starting\_node$ as the two parameters. We store the returned value in a variable named $cost$.

In the *recursive* function, we have the base case when all the bits of $mask$ is set. It means that our garbage vehicle has visited all the bins it was assigned to and now it should go back to its starting place. That is why we return the distance between $last$ and 0 to ensure that the truck is going back to its starting place.

Since this is a DP function, we ought to memorize the already calculated values for different states. So, if we encounter some previously calculated values, we need to return it immediately so that the running time can be sped up.

After that, we can see a ret variable with a large value assigned to it. This ret will store the best value for our current state.

So we are in a state where we have a combination of already visited nodes and a set of unvisited nodes. We can go to any of the unvisited nodes and try to improve our answer. We also have a $last$ node where we visited the last time. Since we have a set of options to go from $last$ to some next node, we have to go to each one of them and find out if we get a better minimum cost.

This is what we do in the next part. In our for loop, we check each of the bits and if the ith bit is unvisited, we visit it by setting the ith bit of $mask$ to 1 and $last$ visited node to ith node. Then we call the function recursively and add the cost of visiting from $last$ to i. In ret, we keep the minimum of them.

Before returning, we have to memorize the values for state $(mask, last)$ to ensure the speed up by DP.

Note that, $distance(i, j)$ means the shortest distance between the two nodes i and j. So, it is obligatory for us to calculate the shortest path cost between each pair of nodes using some *all pair shortest path algorithm*. For instance, we can use **Floyd Warshall APSP Algorithm** which has a complexity of $\mathcal{O}(n^3)$.

So, what we have done in this DP approach is that we have tried to find the best value for the combination of already visited nodes with a last visited node.

The complexity of this approach is $\mathcal{O}(2^n n^2)$. This is simply because we have 2^n combination of $mask$ possible and for each case we have n last nodes.

The complexity is exponential. This DP approach is applicable to cases when the number of garbage bins is not more than 20 to 22.

An implementation of the above idea written in *C++* is provided in the appendix section.

Algorithm 1 DP approach for garbage collection routing

```
FUNCTION DP ($mask, last$)
  if number of 1 bit in $mask$ = n then
    return 0
  end if
  if value for state $(mask, curr)$ already calculated then
    return value for $(mask, curr)$
  end if
  $ret \leftarrow 0$
  for $i = 0$ to $n$ do
    if $i$th bit of $mask$ equals 0 then
      $newmask \leftarrow mask|(1 << i)$
      $curr \leftarrow i$
      $out \leftarrow distance(last, i) + PROCDP(newmask, curr)$
      $ret \leftarrow min(ret, out)$
    end if
  end for
  memorize $ret$ for state $(mask, last)$
  return $ret$
FUNCTION MAIN
Input
  $n \leftarrow$ number of nodes
  $m \leftarrow$ number of edges
  for $i = 1$ to $m$ do
    $E.u \leftarrow$ starting node
    $E.v \leftarrow$ ending node
    $E.w \leftarrow$ cost of edge
  end for
  $mask \leftarrow 1$
  $start\_node \leftarrow 0$
  $cost \leftarrow PROCDP(mask, starting\_node)$
Output $cost$
```

3.4 Approach Using Heuristics

So if we have more than roughly 22 nodes in our garbage bin map, it is very likely that we need to look for better solutions from the perspective of execution time. And here comes the bottleneck. We cannot have any of optimal solution with any non-exponential algorithm.

Over the years, many *heuristics* have been invented or improved keeping TSP in mind. During our work on *garbage vehicle routing*, we considered few of the heuristics to see their performances. In this section, we will discuss them. But before that, we define a particular lower bound defined for TSP.

3.4.1 The Held-Karp Lower Bound

This idea of HK lower bound is important to do a comparison among the heuristics we are going to discuss in subsequent subsections. This lower bound is a common way to measure the performance of TSP heuristics. This is actually the solution to the *linear programming* relaxation of the integer programming formulation of the TSP. The solution can be found in polynomial time by using the Simplex method and a polynomial constraint-separation algorithm [28].

More interestingly, according to [28], HK lower bound averages about 0.8% below the optimal tour length. But its guaranteed lower bound is only $2/3$ of the optimal tour.

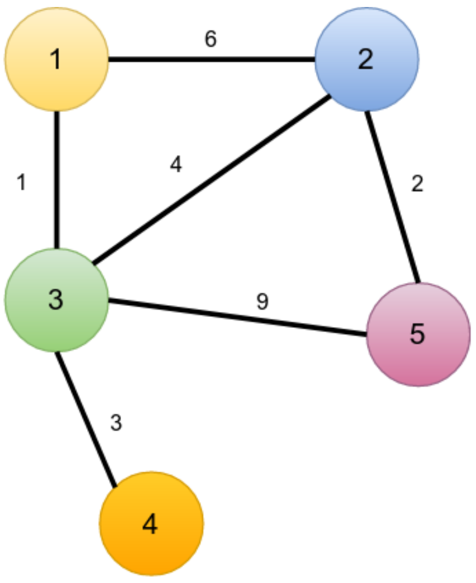

Figure 3.3: *Failure may occur very easily in Nearest Neighbor Heuristic.*

3.4.2 Nearest Neighbor Heuristic

The simplest heuristic that we used was the well known *Nearest Neighbor Heuristic*. This is very simple to understand and easy to implement. Algorithm 2 gives an idea of the heuristic we are currently considering. For the sake of simplicity, we have only provided the main steps of the algorithm.

The complexity of this heuristic is $\mathcal{O}(n^2)$.

The performance of this heuristic is as bad as the simplicity of its idea. Implementing this in real code takes almost no time but it can sometimes miss shorter routes which are easily noticed with human insight, due to its *greedy* nature. Generally, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable. On the other hand, if they are much greater, then it is likely that there are much better tours.

Algorithm 2 Nearest Neighbor Heuristic for grabage collection routing with complexity $\mathcal{O}(n^2)$

```
$c \leftarrow$ a random city selected
while all the nodes are not visited do
    $u \leftarrow$ the nearest unvisited city from $c$
    mark $u$ visited
    $c \leftarrow u$
end while
return to the starting city
```

The bottleneck of this approach is quite clear: its uncertainty of even reaching a feasible tour. As it is sometimes possible that there are few reasonable tours for the garbage trucks but this heuristic finds out the most in-feasible one. An example of a failure case is shown in Figure 3.3. If the node 2 is selected and then we visit node 5, then we have to visit the edge with cost 9 which is violating any near optimal solution.

In general, this heuristic tends to keep its tours within 25% of the Held-Karp lower bound [29].

3.4.3 Double MST Heuristic

Most heuristics of TSP can guarantee a worst-case ratio of 2 which is also known as the *approximation ratio*. What it means is that in the worst case, the tour cost found by this heuristic will be at most twice the optimal tour cost. One of such heuristics is *Double MST* heuristic. The pseudocode for this approach can be found in Algorithm 3. We have also provided a working *C++* code in the appendix.

We need to note the one very important criterion to apply this heuristic which is known as *triangle inequality*. The cost function of the graph that the heuristic will work on has to follow this particular property. This is similar to what its name suggests.

Suppose, we have a graph $G(V, E)$ where the cost function $w(x, y)$ denotes the cost between the nodes x and y. Then according to the *triangle inequality* property:

$w_{ij} \leq w_{ik} + w_{kj}$

for any three vertices of the graph i, j and k.

This particular instance of TSP following the *triangle inequality* is known as $Metric\,TSP$. We will have to convert our given graph in such a way so that it follows the *triangle inequality*. Or in other words, we need to convert our TSP into a $Metric\,TSP$.

In this heuristic, we assume that the graph representing the map of the garbage bins satisfy the *triangle inequality* for its cost function.

The reason behind the name *Double MST* heuristic is that instead of taking the *preorder* traver-

Algorithm 3 Double MST Heuristic for garbage collection routing. This algorithm has a complexity of $\mathcal{O}(n^2 log_2(n)$
.

```
INPUT $G(V, E)$ which follows triangle inequality
  calculate $distance(i, j)$ for all pairs of nodes $i$ and $j$
  $mst \leftarrow KRUSKAL(G)$
  $preOrd \leftarrow$ preorder walk on $mst$
  traverse the nodes in $preOrd$ and take the total $cost$
  return cost
```

sal, we could duplicate each edge to create an *Euler Circuit* and traverse this circuit. Whenever we encountered an already visited node, we simply avoid it and take a shortcut to the next unvisited node. This would do the same thing.

Following three figures demonstrate the idea described in Algorithm 3.

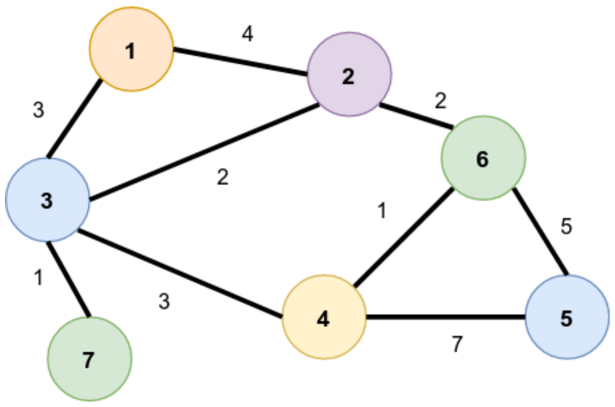

Figure 3.4: *A sample graph.*

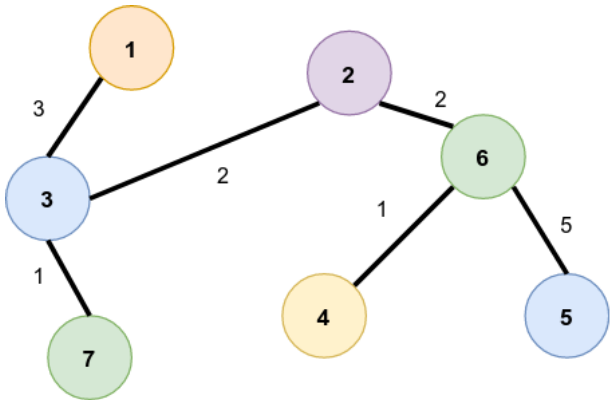

Figure 3.5: MST *of the graph in Figure 3.4*

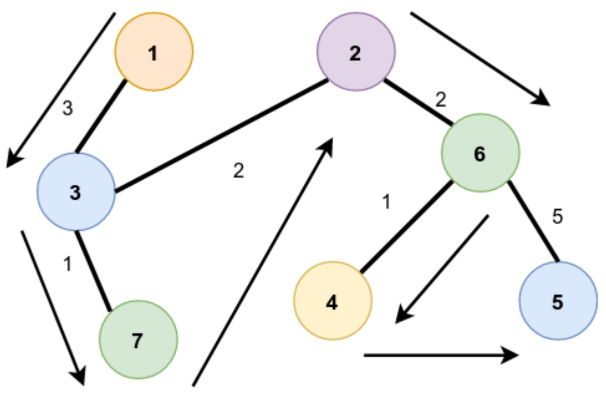

Figure 3.6: *Tour constructed by Double MST heuristic. The arrow lines depict the tour.*

Lastly, comparing with *HK lower bound*, this heuristic generally returns tours within $15 - 20\%$ within this bound.

3.4.4 Christofide's Heuristic

Professor Nicos Christofide [30] extended one of these algorithms and concluded that the worst-case ratio of that extended algorithm was $3/2$. This algorithm is commonly known as Christofide's heuristic [31].

This particular heuristic is a key concept regarding the approach we are trying to build up. But before that, we need to find out a way to reduce a general TSP to a $Metric\, TSP$.

To do this particular reduction, we can actually follow two of the following methods:

Reconstructing the Input Graph

In this method, we run an *APSP* [32] algorithm on the input graph. Floyd-Warshall algorithm can be a choice in this regard. After running this algorithm on the input graph, we now have the shortest cost between each pair of the nodes. We construct a new graph by adding an edge between each pair of nodes with the cost equal the shortest path between them. This process is illustrated in Figure 3.7.

Now it is very easy to see that the above constructed graph follows triangle inequality that we mentioned before. Suppose, we have three nodes i, j and k and the costs of edge between the nodes in the constructed graph are c_{ij}, c_{jk} and c_{ik}. According to triangle inequality, $c_{ij} + c_{jk} \geq c_{ik}$.

If this was not true, then c_{ik} would not even be the edge cost between i and k as there is already a better path existing: $c_{ij} + c_{jk}$. Hence, this newly constructed graph follows the triangle inequality.

Adding an Offset to Each Edge

The plus point of this method is that we actually do not need to reconstruct the graph or find shortest path beforehand (though we will need the shortest path eventually).

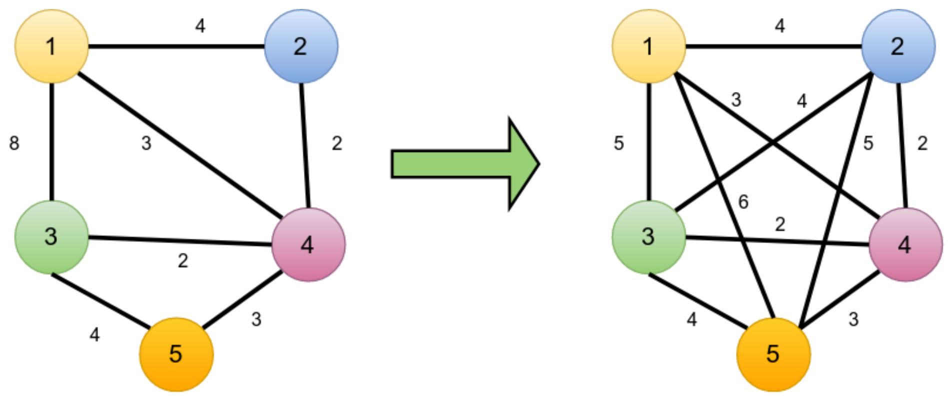

Figure 3.7: *Reconstruction of input graph based on APSP algorithm.*

Let us assume that $C(i, j)$ denotes the cost between the nodes i and j. Now, we find out the maximum such cost $M = \max_{i,j} C(i, j)$. We now define the $Metric\,TSP$ distance $C'(i, j) = C(i, j) + M$.

We need to prove that this change in edge cost can actually reduce the TSP into a $Metric$ TSP.

Let us consider three nodes of our graph: i, j and k. Now,

$C'(i, j) + C'(j, k) = C(i, j) + C(j, k) + 2M$
Again, $C(i, j) + C(j, k) + 2M \geq 2M \geq C(i, k) + M = C'(i, k)$

Since any tour consists of exactly n edges, the transformation adds exactly $n * M$ to any tour.

So, we have found out a method of reducing any given undirected graph into a graph satisfying the triangle inequality. We can now apply the heuristic onto our input graph.

Instead of using $KRUSKAL's$ algorithm, we will use $PRIM's$ algorithm for finding out the MST of $G(V, E)$. It is because the reconstructed graph is dense (edges between each pair of nodes). The complexity of $PRIM$ is $\mathcal{O}(n^2)$.

Before approaching further, let us note that a perfect matching is always possible because any connected graph has an even number of odd degree nodes. We show this well known proof below:

Suppose in a connected graph $G(V, E)$, we have vertices $v_1, v_2, v_3, ..., v_n$ and number of edges is e. Now, if we take the summation of *degrees* of each of the n nodes, then,

Algorithm 4 Christofide's Heuristic for garbage collection routing. This algorithm has a complexity of $\mathcal{O}(n^3)$

.

```
INPUT $G(V, E)$
  calculate $distance(i, j)$ for all pairs of nodes $i$ and $j$
  reconstruct $G(V, E)$, so it follows *triangle inequality*
  $mst \leftarrow PRIM(G)$
  $odds \leftarrow$ set of nodes having an odd degree in $mst$
  $mwm \leftarrow$ *minimum-weight matching* on $odds$
  add $mst$ and $mwm$ together
  create and *Euler Circuit* from the combined graph and traverse it taking shortcuts to avoid
  visited nodes
  $cost \leftarrow$ add costs of the visits
  **return** cost
```

$$\sum_{i=1}^{n} degree(v_i) = 2e$$

Let us assume that the degrees of the first r vertices be even and the remaining $(n - r)$ vertices have odd degree.

Of course,

$$\sum_{i=1}^{r} degree(v_i) = even$$

So,

$$\sum_{i=1}^{n} degree(v_i) = \sum_{i=1}^{r} degree(v_i) + \sum_{i=r+1}^{n} degree(v_i)$$

$$\implies \sum_{i=1}^{n} degree(v_i) - \sum_{i=1}^{r} degree(v_i) = \sum_{i=r+1}^{n} degree(v_i)$$

Now, the left side of the equation above has a value of $even$. So, the right side has to have an $even$ value.

In other words, $(n - r)$ is even which means we can always find a *Minimum Weight Matching*. This matching algorithm is responsible for the $\mathcal{O}(n^3)$ algorithm. We have implemented Christofide's Algorithm in *C++*. Code is attached to the appendix. We also provide the method of Christofide in figures 3.8 and 3.9.

Lastly, *Christofide's Heuristic* tends to keep itself around 10% above the *HK lower bound.* This is a good improvement given the previous heuristics.

So we approached with DP and then with several heuristics to our garbage collection routing

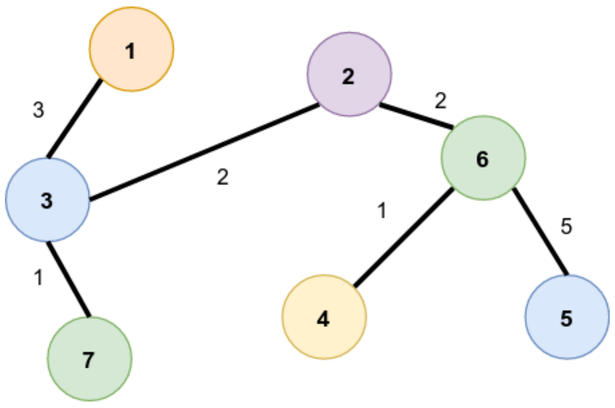

Figure 3.8: *MST of the graph in Figure 3.4*

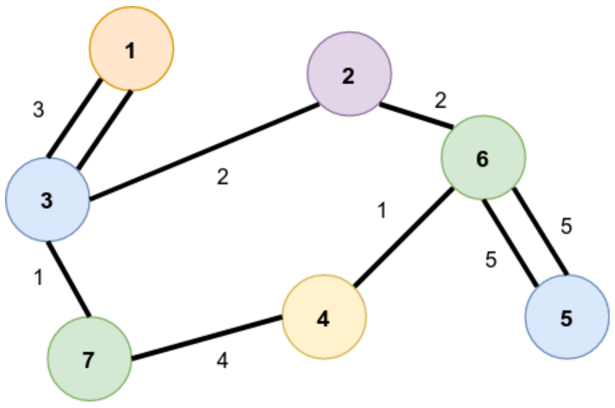

Figure 3.9: *MST and MWM combined.*

problem. The next chapter shows the experimental results on various datasets.

Chapter 4

Experimental Results

In this chapter, we show the experimental results based on randomized data. We show the output comparison between the heuristics we have discussed in the previous chapter. Before approaching to the comparison, we light upon the method of generating data.

4.1 Randomized Graph Generation

We run our codes on random data generated using the code provided in appendix *A.4*. Since in *C++11* the standard library includes `random` which is helpful to generate good random numbers.

Since, we needed a connected graph, we first ensured that each of the nodes is added with an edge with some other node. After that we used randomization to generate more edges and their costs.

Algorithm 5 Random graph generation using *C++11*.

```
.
INPUT
  $n \leftarrow$ number of nodes
  $m \leftarrow$ number of edges
  generate edges to ensure the graph is connected using randomization
  generate the rest edges using randomization
OUTPUT  output $G(V, E)$
```

4.1.1 Sample Run

In this section, we show the comparison between the DP approach and Christofide's heuristic. We generated random data and in the table below, we show ten of them.

Table 4.1: Comparison of performance between the two major approaches desribed in previous chapter.

			Output Cost		Execution Time	
Dataset	N	M	DP	Christofide	DP	Christofide
1	4	6	1706	1706	108.389	0.348
2	5	7	18	18	102.061	0.313
3	7	9	25	34	109.808	0.379
4	10	20	4594	5906	196.229	0.489
5	10	19	5551	5594	99.273	0.47
6	15	20	10774	13003	137.178	0.688
7	15	90	1598	2064	107.119	0.883
8	20	50	7424	7806	1850.22	0.931
9	21	78	3850	5021	5449	1.02
10	50	100	-	22699	-	4.475

We have plotted the data using *Python*. The graphs have been provided in the next page.

4.2 Findings

We generated random graphs on which we have applied our approaches. The table 4.1 shows that DP is always giving the best value but in every case it is taking much more time than the Christofide's heuristic. On the other hand, the heuristic takes a lot less execution time but its output cost is always worse than the DP method.

We know in case of garbage collection, there will be a number of trucks assigned to a number of garbage bins. We are trying to optimize the route after this assignment has been made.

If a truck is assigned to no more than 20 22 bins, we can use the *bitmask DP* approach which will simply provide the best possible cost. On the other hand, if it is assigned with more bins than this limit, we will apply Cristofide's heuristic.

There are various methods to improve the tour found by the heuristic. We had an idea using Genetic Algorithm which we could not implement.

After we have built the tour, we can run a GA on the tour to find out if a better tour possible. For the fitness function, we can use the cost found by the heuristic as a target cost and we will try to even minimize it.

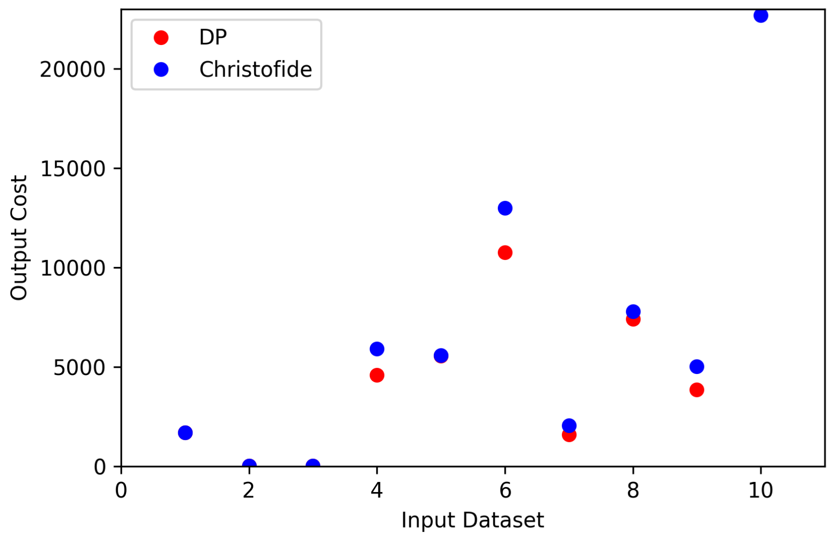

Figure 4.1: *Output cost plot for the two cases.*

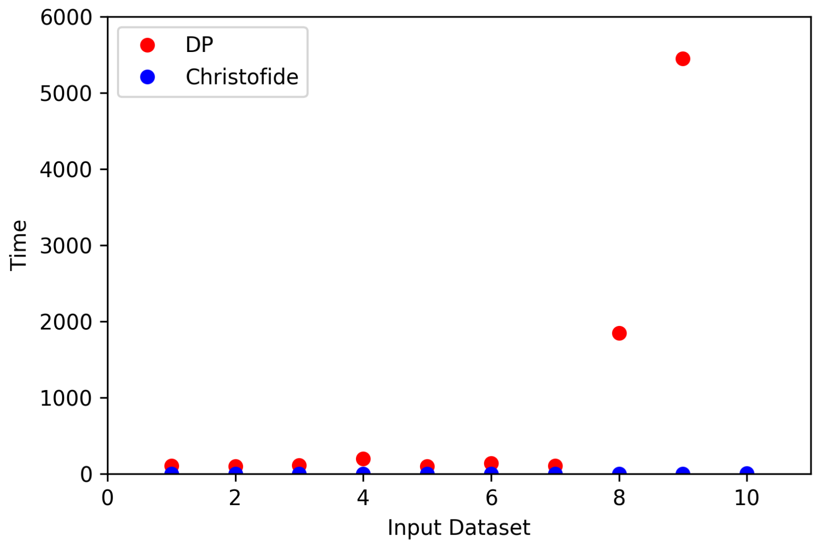

Figure 4.2: *Execution time plot for the two cases.*

Chapter 5

Conclusion

In the beginning of this thesis paper, we have tried to strongly prove the necessity of a standard waste management system for the benefits of a country. And this management system has to be accompanied by a strong waste collection methodology. Since the whole waste management is a huge system from the perspective of any country, a small improvement in the collection measures can actually contribute a huge amount in the long run.

We have also mentioned couple of times that we are looking at this problem from the perspective of Dhaka, the capital city of Bangladesh. Now, the moment we thought about routing garbage trucks in Dhaka city in an optimal way, it was obvious to look for a map of the garbage bins located in the city. Very unfortunately, we have failed to collect the corresponding map. We knocked the corresponding authority several times so that they could provide us a garbage bins location map but apparently, the could not. It is very likely that they do not have such a map.

5.1 Future Work

We have roughly considered the cost of traveling from one bin to another but there can be different kinds of features in costs like, *fuel consumption of garbage trucks, time taken to travel from one bin to another, amount of time spent working on a garbage bin, road blockade or traffic jam in particular roads* etc. It is very likely that the cost function will be much more complex. There is a scope to consider the cost function in a more elegant way to resemble the practical scenario.

There can be other variations of the same problem. For example, it can happen that the truck which has been assigned with some amount of nodes, will not have to go to each of the bins as some of them are not full enough. It is always important in garbage collection to prioritize the bins which are overflown or almost full.

Another popular form of this problem is *garbage collection with time windows.* In this particular variation, garbage trucks have a particular time window within which it has to finish its assigned task. Many papers have been published approaching this variation of the problem.

So we can combine a more complex cost function, a variation of number of bins for a truck and a time window with our approach using DP and Cristofide's algorithm. This can significantly improve our performance.

Another important point is that we actually need a lot of field work. A garbage collection map is a must need for us to apply our approach in a large scale. It is possible to make a small map manually within months so that we can apply our method and run experiments but eventually we will need a fully functioning map to identify the bins and finding the optimal route for them.

There are also scopes of improvement from the perspective of algorithmic performance. One important idea that we could not bring to light in this thesis book is that after finding the tour provided by Christofide's algorithm, we can apply Genetic Algorithm on this tour to find out if a better route is possible. A GA along with a well designed fitness function based on the tour cost found by the heuristic approach can be a good way to expect tour improvement even after we have constructed a satisfactory solution.

5.2 The Last Words

In our thesis, we have proposed the use of DP and Cristofide's heuristic in order to solve the garbage collection routing problem. We have shown the concern regarding Dhaka city.

There are scopes where we can improve our method. One important note is that we have done our experiments based on randomly generated graphs. With the help of real data of the Dhaka city, we have chances to improve our methodology.

References

[1] D. Hoornweg and P. Bhada-Tata, “What a Waste: A Global Review of Solid Waste Management,” tech. rep., World Bank, 2012.

[2] “The Study on Solid Waste Management in Dhaka City,” tech. rep., Dhaka City Corporation and Japan International Cooperation Agency, 2005.

[3] T. Khan, “Starting from the Beginning,” tech. rep., The Daily Star: Star Weekend Magazine, 2012.

[4] A. H. Mahmud, “DNCCs Dumpsters Are a Public Nuisance,” tech. rep., Dhaka Tribune, 2013.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[6] S. A. Hassan, N. G. M. Jameel, and B. ekerolu, “Smart Solid Waste Monitoring and Collection System,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, no. 10, 2016.

[7] M. Hasan and S. M. S. Iqbal, “State of Urban Solid Waste Administration: A GIS Based Analysis of Dhaka South City Corporation (DSCC),” *GEOGRAFIA Online TM Malaysian Journal of Society and Space*, vol. 11, no. 13, 2015.

[8] S. Carl, *Pale Blue Dot: A Vision of the Human Future in Space*. Random House, 1994.

[9] “Solid Waste Management in Dhaka,” tech. rep., Climate and Clean Air Coalition Municipal Solid Waste Initiative, USA., 2014.

[10] “Solid Waste Management: Issues and Challenges in Asia,” tech. rep., Asian Productivity Organization, Tokyo, Japan, 2007.

[11] “ICS 161: Design and Analysis of Algorithms Lecture notes for March 12, 1996.” https://www.ics.uci.edu/~eppstein/161/960312.html, 1996.

[12] R. Matai, S. P. Singh, and M. L. Mittal, “Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches,” 2010.

[13] M. Grotschel, M. Junger, and G. Reinelt, “Optimal Control of Plotting and Drilling Machines: A Case Study,” *Mathematical Methods of Operations Research*, vol. 35, no. 1, pp. 61–84, 1991.

[14] R. D. Plante, T. J. Lowe, and R. Chandrasekaran, “The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristics,” *Operations Research*, vol. 35, pp. 772–783, 1987.

[15] R. E. Bland and D. E. Shallcross, “Large Traveling Salesman Problem Arising from Experiments in X-ray Crystallography: A Preliminary Report on Computation,” *Operations Research Letters*, vol. 8, no. 3, pp. 125–128, 1989.

[16] J. K. Lenstar and A. H. G. Rinooy Kan, “Some Simple Applications of the Travelling Salesman Problem,” *Stichting Mathematisch Centrum*, 1974.

[17] H. D. Ratliff and A. S. Rosenthal, “Order-Picking in a Rectangular Warehouse: A Solvable Case for the Travelling Salesman Problem,” *Operations Research*, vol. 31, pp. 507–521, 1983.

[18] I. Mariusz and J. Marianna, “An Approach to the Waste Collection Routing Problem in the Municipal Services Companies,” *PRACE NAUKOWE POLITECHNIKI WARSZAWSKIEJ*, 2015.

[19] A. Katkar, “Improvement of Solid Waste Collection by Using Optimization Technique,” *International Journal of Multidisciplinary Research*, vol. 2, no. 4, 2012.

[20] V. K. Nikolaos, P. Katerina, and G. L. Vassili, “Genetic Algorithms for Municipal Solid Waste Collection and Routing Optimization,” *Thesis of School of Electrical and Computer Engineering, School of Electrical and Computer Engineering*, 2012.

[21] V. P. Ingo and R. A. Adel, “Optimal Routing for Solid Waste Collection in Cities by Using Real Genetic Algorithm,” *Ingenierurtechnik Merck KGaA.*, 2011.

[22] A. Mahmuda, M. A. Hannan, and H. Basri, “Particle Swarm Optimization Modeling for Solid Waste Collection Problem with Constraints,” *IEEE 3rd International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, 2015.

[23] H. J. Van, F. Shahin, and Z. Arnavut, “Application of Particle Swarm Optimization for Traveling Salesman Problem to Lossless Compression of Color Palette Images,” *IEEE 3rd International Conference on System of Systems Engineering*, pp. 1–5, 2008.

[24] M. Bogna, “Route Planning of Separate Waste Collection on a Small Settlement,” *Problemy Transportu*, vol. 9, no. 1, 2014.

[25] S. Anna, R. Joel, and G. Andre, "Simulation-based Optimization of a Real-world Travelling Salesman Problem Using an Evolutionary Algorithm with a Repair Function," *International Journal of Artificial Intelligence and Expert Systems (IJAE)*, vol. 6, no. 3, 2015.

[26] R. Remigiusz and S. Zbigniew, "Solving the Problem of Vehicle Routing by Evolutionary Algorithm," *Advances in Science and Technology Research Journal*, vol. 10, no. 29, 2016.

[27] V. K. Nikolaos, N. Doukas, K. Maria, and D. Gerasimoula, "Routing Optimization Heuristics Algorithms for Urban Solid Waste Transportation Management," *WSEAS TRANSACTIONS on COMPUTERS*, vol. 7, no. 12, 2008.

[28] D. S. Johnson, L. A. McGeoch, and E. E. Rothberg, "Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound," *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 341–350, 1996.

[29] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," 1995.

[30] "Nicos Christofides." `https://de.wikipedia.org/wiki/Nicos_Christofides`.

[31] "Christofide's Algorithm." `https://en.wikipedia.org/wiki/Christofides_algorithm`.

[32] "Floyd-Warshall Algorithm." `https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm`.

Appendix A

Codes

A.1 Implementation of DP Approach

Below is a *C++* implementation of the method explain in *section 3.3*.

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

#define ms(x,y) memset (x, y, sizeof (x))
#define INF 100000000
#define MAX 20
#define FOR(i,a,b) for (int i=(a); i<(b); i++)
#define FORr(i,a,b) for (int i=(a); i>=b; i--)

ll cost[MAX][MAX], memo[1<<MAX][MAX];
int n, m, u, v, w, lastpos;

void FloydWarshall()
{
    FOR(i,0,n)
    {
        FOR(j,0,n)
        {
            FOR(k,0,n)
            {
                    cost[j][k]=min(cost[j][k],cost[j][i]+cost[i][k]);
                }
            }
        }
}

ll call(int mask, int last)
{
    if(mask==((1<<n)-1))
    {
        // returning to the base
        return cost[last][0];
    }

    // memorization
    if(memo[mask][last]!=-1) return memo[mask][last];

    ll ret=INF;

    for(int i=0; i<n; i++)
    {
        if(!(mask & (1<<i)))
        {
            int newmask=(mask|(1<<i));
            int newlast=i;

            // calling next states
            ll out=cost[last][i]+call(newmask,newlast);

            ret=min(ret,out);
        }
    }

    return memo[mask][last]=ret;
}

int main()
{
    // Nodes are indexed from 0
    // Truck at node 0

    cin>>n>>m;

    // Initializing costs
    FOR(i,0,n)
    {
        FOR(j,0,n)
        {
            if(i==j) cost[i][j]=0;
            else cost[i][j]=INF;
        }
    }

    FOR(i,0,m)
    {
        cin>>u>>v>>w;
        u--, v--; // ensuring 0 based indexing
        cost[u][v]=cost[v][u]=w;
    }

    FloydWarshall();

    ms(memo,-1); // clearing memory

    int mask=1, start_node=0;

    ll ans=call(mask,start_node);

    cout<<"Cost:␣"<<ans<<endl;

    return 0;
}
```

A.2 Double MST or 2-Approximation Algorithm Implementation

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair <int, int> pii;
typedef vector<pair<int,int> > vpii;

#define MP make_pair
#define ms(x,y) memset (x, y, sizeof (x))
#define pb push_back
#define INF 1000000000000LL
#define MAX 100
#define debug cout<<"A"<<"\n"
#define prnt(a) cout<<a<<"\n"
#define FOR(i,a,b) for (int i=(a); i<(b); i++)
#define FORr(i,a,b) for (int i=(a); i>=b; i--)
#define VecPrnt(v) FOR(J,0,v.size()) cout<<v[J]<<"␣"; cout<<endl
#define PrintPair(x) cout<<x.first<<"␣"<<x.second<<endl

struct info
{
        int u, v;
        double w;

        info(){}

        info(int u, int v, double w) : u(u), v(v), w(w) { }
};


vector<info> edges;
vector<int> graph[MAX];


class Approximation
{
        int parent[MAX];
        int n;
```

```
	double dist[MAX][MAX];

	void dfs(int u, int p)
	{
		preorder.pb(u);

		FOR(j,0,graph[u].size())
		{
			int v=graph[u][j];
			if(v==p) continue;

			dfs(v,u);
		}
	}

	static bool comp(info &a, info &b)
	{
		return a.w<b.w;
	}

	double distance(pii a, pii b)
	{
		ll ret=(a.first-b.first)*(a.first-b.first)+(a.second-b.se
		return sqrt((double)ret);
	}

	void calcDistance(vpii &all)
	{
		FOR(i,0,all.size()-1)
		{
			FOR(j,i+1,all.size())
			{
				double d=distance(all[i],all[j]);
				edges.pb(info(i,j,d));
				dist[i][j]=dist[j][i]=d;
			}
		}
	}

```

```
        int find(int r)
        {
                if(parent[r]==r) return r;
                return parent[r]=find(parent[r]);
        }

        double Kruskal()
        {
                sort(edges.begin(),edges.end(),comp);

                int cnt=0;

                FOR(i,0,(int)edges.size())
                {
                        int u=find(edges[i].u);
                        int v=find(edges[i].v);

                        if(u!=v)
                        {
                                parent[u]=v;
                                cnt++;

                                graph[edges[i].u].pb(edges[i].v);
                                graph[edges[i].v].pb(edges[i].u);

                                if(cnt==n-1) break;
                        }
                }
        }

public:

        vector<int> preorder;

        double Walk()
        {
                double tour=0;
                dfs(0,-1);
                preorder.pb(0);

                FOR(i,0,preorder.size()-1)
                {
                        tour+=dist[preorder[i]][preorder[i+1]];
                }

                return tour;
        }


        Approximation(int cities)
        {
                FOR(i,0,cities)
                {
                        parent[i]=i;
                        edges.clear();
                }
                n=cities;
                preorder.clear();
        }

        double getAnswer(vpii &all)
        {
                calcDistance(all);
                Kruskal();
                return Walk();
        }
};


int main()
{
        int n;
        vector<pii> Points;

    cin>>n;

    Points.resize(n);

    FOR(i,0,n)
    {
        cin>>Points[i].first>>Points[i].second;
    }

    Approximation tsp(n);
    cout<<"Approximate␣cost:␣"; prnt(tsp.getAnswer(Points));
    cout<<"Path:"<<endl;

    // VecPrnt(tsp.preorder);

    FOR(i,0,tsp.preorder.size())
    {
        cout<<Points[tsp.preorder[i]].first<<"␣"<<Points[tsp.preorder[i]]
    }

    return 0;
}
```

A.3 Christofide's Heuristic Implementation

```
#include <bits/stdc++.h>
using namespace std;

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define ll long long
#define MAX 100
#define INF 1e7
#define FOR(i,a,b) for (int i=(a); i<(b); i++)
#define FORr(i,a,b) for (int i=(a); i>=b; i--)
#define pb push_back
#define VecPrnt(v) FOR(J,0,v.size()) cout<<v[J]<<"␣"; cout<<endl
```

```
#define ms(x,y) memset (x, y, sizeof (x))

int n, m, cost[MAX][MAX], prv[MAX]; bool visited[MAX];
ll d[MAX];
VI mst[MAX], match[MAX], odds;
VPII touredges;
bool adjMST[MAX][MAX], done[MAX][MAX];

void FloydWarshall()
{
    FOR(i,0,n)
    {
        FOR(j,0,n)
        {
            FOR(k,0,n)
            {
                cost[j][k]=min(cost[j][k],cost[j][i]+cost[i][k]);
            }
        }
    }
}

T Prim (const VVT &w, VPII &edges)
{
        int n = w.size();
        VI found (n);
        VI prev (n, -1);
        VT dist (n, 1000000000);
        int here = 0;
        dist[here] = 0;

        while (here != -1)
        {
                found[here] = true;
                int best = -1;
                for (int k = 0; k < n; k++)
                {
                        if (!found[k])
                        {
```

```
                                        if (w[here][k] != -1 && dist[k] > w[here]
                                        {
                                                dist[k] = w[here][k];
                                                prev[k] = here;
                                        }
                                        if (best == -1 || dist[k] < dist[best]) b
                                }
                        }
                        here = best;
                }

                T tot_weight = 0;
                for (int i = 0; i < n; i++)
                {
                        if (prev[i] != -1)
                        {
                                edges.push_back (make_pair (prev[i], i));
                                tot_weight += w[prev[i]][i];
                                mst[prev[i]].pb(i);
                                mst[i].pb(prev[i]);
                        }
                }
                return tot_weight;
}

int MinCostMatching(const VVI& cost, VI& Lmate, VI& Rmate)
{
    int n = int(cost.size());

    // construct dual feasible solution
    VI u(n);
    VI v(n);
    for (int i = 0; i < n; i++) {
        u[i] = cost[i][0];
        for (int j = 1; j < n; j++)
            u[i] = min(u[i], cost[i][j]);
    }
    for (int j = 0; j < n; j++) {
        v[j] = cost[0][j] - u[0];
        for (int i = 1; i < n; i++)
            v[j] = min(v[j], cost[i][j] - u[i]);
    }

    // construct primal solution satisfying complementary slackness
    Lmate = VI(n, -1);
    Rmate = VI(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (Rmate[j] != -1)
                continue;
            if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
                Lmate[i] = j;
                Rmate[j] = i;
                mated++;
                break;
            }
        }
    }

    VI dist(n);
    VI dad(n);
    VI seen(n);

    // repeat until primal solution is feasible
    while (mated < n) {

        // find an unmatched left node
        int s = 0;
        while (Lmate[s] != -1)
            s++;

        // initialize Dijkstra
        fill(dad.begin(), dad.end(), -1);
        fill(seen.begin(), seen.end(), 0);
        for (int k = 0; k < n; k++)
            dist[k] = cost[s][k] - u[s] - v[k];

        int j = 0;
        while (true) {

            // find closest
            j = -1;
            for (int k = 0; k < n; k++) {
                if (seen[k])
                    continue;
                if (j == -1 || dist[k] < dist[j])
                    j = k;
            }
            seen[j] = 1;

            // termination condition
            if (Rmate[j] == -1)
                break;

            // relax neighbors
            const int i = Rmate[j];
            for (int k = 0; k < n; k++) {
                if (seen[k])
                    continue;
                const int new_dist = dist[j] + cost[i][k] - u[i] - v[k];
                if (dist[k] > new_dist) {
                    dist[k] = new_dist;
                    dad[k] = j;
                }
            }
        }

        // update dual variables
        for (int k = 0; k < n; k++) {
            if (k == j || !seen[k])
                continue;
            const int i = Rmate[k];
            v[k] += dist[k] - dist[j];
            u[i] -= dist[k] - dist[j];
        }
        u[s] += dist[j];

        // augment along path
        while (dad[j] >= 0) {
            const int d = dad[j];
            Rmate[j] = Rmate[d];
            Lmate[Rmate[j]] = j;
            j = d;
        }
        Rmate[j] = s;
        Lmate[s] = j;

        mated++;
    }

    int value = 0;
    for (int i = 0; i < n; i++)
    {
        if(done[i][Lmate[i]]) continue;

        value += cost[i][Lmate[i]];
        mst[odds[i]].pb(odds[Lmate[i]]);
        mst[odds[Lmate[i]]].pb(odds[i]);

        done[i][Lmate[i]]=done[Lmate[i]][i]=true;
    }

    return value;
}

int dfsCount(int u, bool marked[])
{
        marked[u]=true;
        int cnt=1;

        for(int i=0; i<n; i++)
        {
                if(adjMST[u][i] && !marked[i])
                {
                        cnt+=dfsCount(i,marked);
                }
        }

        return cnt;
}
void removeEdge(int u, int v)
{
        adjMST[u][v]=0;
        adjMST[v][u]=0;
}
void addEdge(int u, int v)
{
        adjMST[u][v]=1;
        adjMST[v][u]=1;
}
bool validNext(int u, int v)
{
        int cnt=0;

        for(int i=0; i<n; i++)
        {
                if(adjMST[u][i]) cnt++;
        }

        if(cnt==1) return true;
        bool marked[n];
        ms(marked,false);
        int cnt1=dfsCount(u,marked);
        removeEdge(u,v);
        ms(marked,false);
        int cnt2=dfsCount(u,marked);
        addEdge(u,v);

        return (cnt1>cnt2) ? false : true;
}

void findEulerTour(int u)
{
        for(int k=0; k<n; k++)
        {
                if(!adjMST[u][k]) continue;
                if(validNext(u,k))
                {
                        touredges.pb({u,k});
                        // cout<<"inserting: "<<u<<" "<<k<<endl;
                        removeEdge(u,k);
                        findEulerTour(k);
                }
        }
}

VI fleury()
{
        findEulerTour(0);
        VI tour; tour.pb(0);

        for(auto it: touredges)
        {
                tour.pb(it.second);
        }
        return tour;
}

int main()
{
        // Nodes are indexed from 0
        // Truck at node 0

        // freopen("in1.txt","r",stdin);

    int st=clock();

        cin>>n>>m;

        // Initializing costs
        FOR(i,0,n)
        {
            FOR(j,0,n)
            {
                if(i==j) cost[i][j]=0;
                else cost[i][j]=INF;
            }
        }

        int u, v, w;

        FOR(i,0,m)
        {
            cin>>u>>v>>w;
            u--, v--; // ensuring 0 based indexing
            cost[u][v]=cost[v][u]=w;
        }

        FloydWarshall();

        VVT W(n,VT(n));
        VPII edges;

        FOR(i,0,n) FOR(j,0,n) W[i][j]=cost[i][j];

        int mstcost=Prim(W,edges);

        // cout<<"MST: "<<mstcost<<endl;

        FOR(i,0,n)
        {
                if(mst[i].size()%2)
                {
                        odds.pb(i);
                }
        }

        VecPrnt(odds);

        VVI C(odds.size());
        VI L, R;

        FOR(i,0,odds.size())
        {
                C[i].resize(odds.size());

                FOR(j,0,odds.size())
                {
                        C[i][j]=cost[odds[i]][odds[j]];
                        if(odds[i]==odds[j]) C[i][j]=1e7;
                }
        }
        int mwmcost=MinCostMatching(C,L,R);
        // cout<<"min-weight-matching: "<<mwmcost<<endl;
        // we have combined the MST and MWM which is an Euler Circuit
        FOR(i,0,n)
        {
                FOR(j,0,mst[i].size())
                {
                        adjMST[i][mst[i][j]]=true;
                }
                // cout<<i<<": "; VecPrnt(mst[i]);
        }

        // Now we visit the euler circuit and get the cost as well as pat
        // We use Fleury's Algorithm in this case.

        VI tour=fleury();

        cout<<"Tour␣Cost:␣"<<mstcost+mwmcost<<endl;

        int en=clock();

        cout<<"Time:␣"<<(en-st)/(double)CLOCKS_PER_SEC*1000<<endl;

        return 0;
}
```

A.4 Random Graph Generation in C++11

```
#include <bits/stdc++.h>
```

```

#define MAX 100
#define MAXCOST 1000
#define prob 0.5

using namespace std;

double getRandom()
{
        random_device rd;
    mt19937 mt(rd());
    uniform_real_distribution<double> dist(0,1.0);

    return dist(mt)*MAX;
}

int getRandomCost()
{
        random_device rd;
    mt19937 mt(rd());
    uniform_real_distribution<double> dist(0,MAXCOST);

    return dist(mt);
}

vector<int> v;
bool chosen[MAX], adj[MAX][MAX];
vector<pair<int,int> > edges;

int main()
{
        int n, m;

        cin>>n>>m;
        cout<<n<<"␣"<<m<<endl;

        while(true)
        {
                bool found=false;
                for(int i=1; i<=n; i++)
                {
                        double d=getRandom();

                        if(d>prob && !chosen[i])
                        {
                                chosen[i]=true;

                                if(v.size())
                                {
                                        found=true;
                                        int d2=(int)getRandom()%v.size();
                                        edges.push_back({v[d2],i});
                                        adj[v[d2]][i]=true;
                                        adj[i][v[d2]]=true;
                                }

                                v.push_back(i);
                        }

                        if(found)
                        {
                                m--;
                                break;
                        }
                }

                if(v.size()==n) break;
        }

        while(m>0)
        {
                int i=(int)getRandom()%n;
                int j=(int)getRandom()%n;

                if(i==j || adj[v[i]][v[j]]) continue;

                edges.push_back({v[i],v[j]});
                m--;
                // cout<<"m: "<<m<<endl;
                adj[v[i]][v[j]]=adj[v[j]][v[i]]=true;
        }


        for(int i=0; i<edges.size(); i++)
        {
                cout<<edges[i].first<<"␣"<<edges[i].second<<"␣"<<getRando
        }

        return 0;
}
```