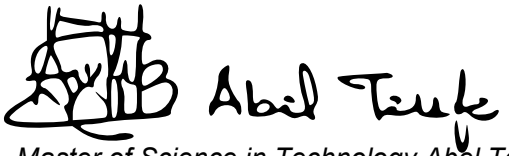
Aalto University
School of Engineering

Route optimization using arc routing for household waste collection

Master's Thesis
Department of Real Estate, Planning and Geoinformatics
School of Engineering
Aalto University

Espoo, *30 November* 2015



Master of Science in Technology Abel Terefe

Supervisor: Professor *Kirsi Virrantaus*
Instructors: Dr. *Jussi Nikander (Natural Resources Institute Finland)*, Dr. *Ari Serkkola (Department of Civil and Environmental Engineering)*

Aalto University
School of Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author	Abel Terefe
Title of thesis	Route optimization using arc routing for household waste collection
Degree programme	Degree Programme in Geomatics
Major	Geoinformation Technology **Code** IA3002
Thesis supervisor	Professor Kirsi Virrantaus
Thesis advisor(s)	Dr. Jussi Nikander, Dr. Ari Serkkola
Date	30.11.2015 **Number of pages** 5+39 **Language** English

Abstract

Arc routing is concerned with the traversal of edges of a graph network. This thesis discusses the directed Chinese postman problem (DCPP), a type of arc routing problem on a directed graph. The Chinese postman problem is concerned with a mailman that starts at the post office, traverses every street in his territory at least once and then returns to the post office. The objective being, while delivering mails, to walk as little as possible. Incidentally, the Chinese postman problem is the 'edge' equivalent of the traveling salesman problem. Unlike the traveling salesman problem however, the chinese postman problem on undirected graphs has polynomial running time algorithm; which makes it an appealing choice for some application domains. In some ways, the DCPP relates to the problem of household waste collection due to the fact that, waste bins are located along street edges and a collection truck essentially needs to traverse streets to get to the bins. A pseudo-polynomial algorithm and its implementation is presented for demonstrating the solution to the DCPP.

Keywords Chinese postman problem, arc routing, routing in household waste collection

Preface

I like to thank Ari Serkkola and the Department of Civil and Environmental Engineering at Aalto University for providing funding and supervision while writing this thesis.

I like to thank Jussi Nikander, *Natural Resources Institute Finland*, who not only advised the thesis patiently, but also helped me obtain materials which proved to be invaluable resources to this thesis. I also like to thank Professor Kirsi Virrantaus and the staff at the Department of Real Estate, Planning and Geoinformatics.

I like to thank my friends: Weiming Wu, Muhammad Usman and Kalle Säilä for their support and advices.

Finally, I like to thank my parents who at all times stood by me.

Espoo, November 30, 2015

Abel Terefe

Contents

Symbols and abbreviations

Symbols

N	Natural number
Z	Integer
R	Real number

Operators

$\dot{\bigcup}$	disjoint union
$\subseteq$	subset
$G \rightsquigarrow F$	Path from G to F
$E \setminus M$	E without M, set difference
$A \oplus B$	symetric difference between A and B
$\sum_i$	sum over index i

Abbreviations

CPP	Chinese postman problem
DCPP	Directed Chinese postman problem
UCPP	Undirected Chinese postman problem
RCPP	Rural Chinese postman problem
CARP	Capacitated Arc Routing Problem
TSP	Travelling Salesman problem

1 Introduction

In any modern town, the collection of household refuse is an important logistical operation. For generations, people have used several methods to manage household refuse. Today, in the developed world, waste collection trucks are used to collect and transport refuse from households and industrial units to recycling centers, power plants and landfills where: it is treated and recycled to yield raw materials for consumer goods, burnt for energy production or buried in a landfill. For example in Finland, in the year 2012 about 66% of the combustible waste was burned to give energy making up about 10% of the overall energy production powering homes and industries. The largest portion of the incinerated waste comes from forestry and forest industries and the second largest portion comes from municipal waste and other mixed waste [32].

Waste collection in cities is a complex and capital intensive process. That is because of the sheer volume of waste containers, even with a medium size city, and the frequency with which containers have to be emptied. This creates economic burden on municipalities whose responsibility includes managing household waste. According to Section 34 of the Waste Act by the Ministry of Environment of Finland, municipalities must ensure:

- The availability of waste transport
- A sufficient number of regional reception points are available for hazardous and non-hazardous waste
- A sufficient amount of diverse forms of waste management services are available such as separate collection in compliance with order of priority
- The collection and transport of waste is organized and scaled to handle the quantity and quality of waste generated as much as possible
- Sufficient information is distributed with suitable frequency on arrangements for waste transport and the regional reception of waste

On the other hand, property owners and housing companies are bound to organize waste collection points and containers for household waste and it is the duty of waste producers to take the waste to these points. The various waste types are collected separately into labeled containers by waste producers. This eventually simplifies the treatment process and also part of the waste has a market value since it can be sold later with little or no treatment.

Often times collection containers are of two type. These are the surface collection containers and deep collection containers. The containers come in different forms and sizes. Surface collection container usually are 140 liters (Biowaste containers), 240 liters, and 600 liters. The containers are color coded to indicate the kind of waste they contain.

Town houses and apartments have their own collection containers for paper, card board, metal and glass. Whereas batteries, carton liquid packaging, and hazardous

waste are collected at a regional collection points. Surface collection containers are often emptied once a week. Deep collection containers, such as Molok^{TM} and Uppo^{TM} are placed partially below the ground with a lifting bag made of a strong textile material inside the container. Deep collection containers are much bigger in size than surface collection containers and as a result, they are often emptied once in two weeks depending on the waste type and regulations [27]. The deep collection containers come in three sizes 1300 litres for biowaste, glass and metal, 3000 litres for paper and cardboard and dry waste containers take up to 5000 litres.

The transportation of waste from collection containers to the final destination is organized by local municipal authorities through agreements with private waste transportation companies. The waste transportation companies operate a fleet of vehicles and are offered contracts through competitive bidding. As a result, the cost of transportation is lowered to a minimum and since transportation services are bought in bulk, municipalities have advantage over other competitors who might like to push prices. It is also possible for contractual agreements to be signed between property owners and transportation companies independent of the authorities. If the municipality organizes the transportation, then it has the upper hand to set prices for property owners. According to Ramboll^{TM} in 2006, this system was used in 33% of the municipalities and 50% of the population whereas independent contractual agreements were made in about 47% of the municipalities and 40%of the population [27].

Transportation is one of the main reasons why municipal waste collection is so expensive. It is essential to optimize transportation for a viable waste management program. In the last decade significant achievements have been made in the area of transportation, logistics and operations researches. With the aid of GIS and the global positioning and navigation systems, it has become possible to precisely plan and execute transportation schemes. The key gains of an automated optimized transportation system is discussed below.

1.1 Vehicle running cost

1. *Reduced fuel consumption* The fuel consumption of a vehicle depends not only on the distance driven but also on the driving pattern of the vehicle, which includes factors such as the speed, acceleration, gear changes, road condition, road gradient, and load and vehicle condition. Fuel price takes up the larger percentage of the cost.

2. *Reduced air pollution* The environmental issues that arise in household waste collection are mainly from the combustion process of the diesel fuel. In addition, environmental pollution can arise, on a minor level, from noise, vehicle oil spill, wear of brakes and tires and other related factors.

3. *Reduced maintenance cost* maintenance costs such as tire, brakes, oil etc will be reduced.

1.2 Working hours

Optimizing routes will help to achieve reduced work hours for drivers. This will in turn help to save spending.

1. *Reduced driver's fatigue and health problem* Driver fatigue and exhaustion while at work is a major issue in transportation. According to a study by the International Road Transport Union in 2006, fatigue and falling asleep is the main cause of approximately 18.6% of single truck accidents [21]. The European Union has enforced a strict regulation on driving hours of trucks which was effective since 2007.

   Such regulations, while protecting road safety, impose heavier burden on trucking companies since they would have to employ more stuff to comply with the legal time windows mentioned in the regulations. However, with the use of a route optimizer, such cost can be significantly reduced.

2. *Reduced overtime* Getting the job done within a given regular time window means reduced overtime payments.

1.3 Service quality

Route optimization will enable to achieve improved service quality. This will be reflected in many ways:

1. *Quicker and efficient collection/pickup* Automated and optimized routing enables a much quicker and efficient service which increases customer satisfaction and reduces overall cost.

2. *Improved prediction system* Route optimization also enables a much reliable prediction system. The prediction system can often be a separate module where it is constantly fed with information from a real-time route optimization system such that vehicles can be rerouted or rescheduled ahead of time with much flexibility.

1.4 Problem definition

Waste bins are located along the streets of a well-defined road network. They are emptied regularly by compactor trucks with finite capacity. Only one type of waste is collected by the truck during a single route. There are multiple types of bins. The collection is done during workdays. The trucks visit the waste disposal site before returning to their depot if they are loaded by more than 50% of their capacity. A work-hour for the trucks is typically 8hrs and if that is exceeded, an overtime payment has to be made. A work-day is typically divided into two by a lunch break that lasts for 30min and hence two tours can be made. In addition, drivers have 15min coffee-break. The vehicles are of different kind and capacity. Emptying the waste bins and unloading at the waste disposal site should also be done within a given time window. The waste collection process tends to follow a periodic cycle of

typically two weeks in residential areas. The period could vary depending on the population density and land use of the area. The amount of solid waste is highly variable and the accumulation of waste depends on several factors such as the number of inhabitants sharing a container, GDP per capita, lifestyle, time of the year, etc [25]. This indicates that the amount of waste is a random variable.

Therefore, the vehicle routing problem in waste collection has the following constraints and characteristics.

- Time windows: *it has to be done within fixed time bounds*
- Limited capacity: *vehicles can only carry a finite amount*
- Periodicity: *the emptying is cyclic, 1 time/week, 2 times/week or more*
- Randomness: *the amount of waste is random* [25]

This leads to the conclusion that the waste collection route optimization is a stochastic periodic capacitated vehicle routing problem with time windows. All these factors add up to complicate the solution for the basic problem. Of course, in this thesis we tackle a much stripped down version of the general problem. Dealing with the general problem in its full glory is beyond the scope of this thesis.

1.5 Research questions

The goal of this thesis is to find a set of algorithms that can be used to solve the problem of optimizing routes in household waste collection. In particular, it studies the behavior of cyclic routes where a waste collecting truck drives across streets and gets back to its starting position. The status quo is that there is only a batch of waste points that are known to be emptied on a given day. There is no route optimization solution being used and the entire routing matter is left to the driver of the truck as I learned during interviews. Some of the relevant questions in this research are:

- What is the behavior of the route optimization problem in household waste collection? This includes, the study of the physical constraints that the problem is subjected to.
- What are the most important elements that affects the overall cost of a route/cycle? This is important to identify because not all constraints affect the cost equally and therefore, when simplification of the solution is needed we know which constraints to drop from our model, without affecting the overall quality of the final solution.
- What alternative routing models can be used household waste collection? There are several models in VRP and each with its own merit and demerit. It is essentially the nature of the VRP problem, that there exists no general solution that is good in all circumstances. And therefore, it is worthwhile studying specific models that are tailored to tackle a certain group of problem, in this case, household waste collection.
- How good is the final model compared to preexisting models? This is concerned with analytical comparison on the performance of a model.

2 Materials and methods

Vehicle routing is divided into two major categories. These are arc routing problems (ARP) and node routing problems (NRP). In general, node routing problems are referred to as VRP and arc routing problems as ARP. However, it is important to distinctly identify the two, as there is a fundamental difference in how the problems are defined, solved and applied.

2.1 Vehicle routing problem

Node routing is a type of vehicle routing problem concerned with finding a closed route or a cycle connecting every vertex of a graph $G = (V, E \cup A)$. In node routing, the purpose is to find a minimum cost route connecting every node (vertex[1]). This relates to the classical traveling salesman problem. The vehicle routing problem is one of the earliest of its kind. The first such paper was the *truck dispatching problem* [7] which is concerned with the optimum routing of a fleet of gasoline delivery trucks between bulk terminal and a large number of service stations.

2.2 Arc routing problem

Arc routing is another kind of vehicle routing problem that is concerned with finding a tour that visits every edge exactly once. In arc routing, the purpose is to find a minimum cost tour that consists of every edge and returns to the origin vertex. It turns out that, it is not possible to find a tour with no repeating edges for all kinds of graphs. It is only possible to do that on a special kind of graph called *eulerian graph*, named after the famous Swiss Mathematician Leonard Euler who is considered by many as the father of graph theory. The question Euler faced was whether there is a marching band route starting on an island in a city called Königsberg, which would traverse each of the cities seven bridges exactly once with no repetitions and end on the island from which the route started. Euler proved that there is no such route marking the beginning of graph theory as we know it today [24].

Figure 2 shows the graph representation of the city of Königsberg. The edges represent the bridges and the vertexes represent the land. The island of Kneiphof is represented by the vertex B. An eulerian tour on such graph is impossible because the graph is not an even graph. An even graph is a graph that has vertexes with even degrees. The degree of a vertex is the number of edges connected to it. In the next chapters we will discuss more on eulerian graphs.

In waste collection, the vehicle routing problem can be studied, in principle, both as a node routing problem and arc routing problem. However various researches in the area of household waste collection, have indicated that the problem of route optimization is an arc routing problem. The suitability of the two methods for such problems depends on the scope of the problem. Arc routing is a reasonable choice in densely populated areas, typically in cities. This is because in cities we may find

[1]We use the terms *vertex* and *node* interchangeably to mean the same thing through out this paper.

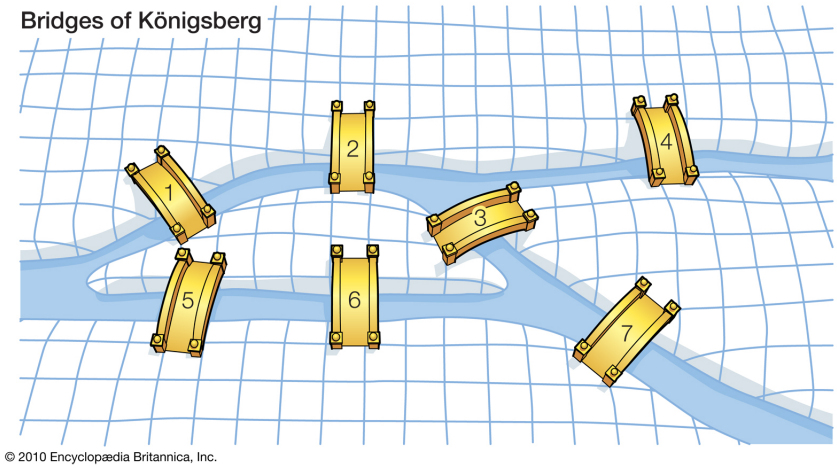


Figure 1: The bridges of Königsberg [12]

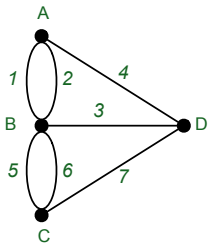


Figure 2: Graph representation of the bridges of Königsberg

a number of waste bins along a single street and it would be a reasonable thing to compute the streets instead of the individual waste bins/containers. In arc routing, the nodes are the street intersections and the edges are the street segments. The goal will be to find a tour where each edge is visited at least once. The edges that don't contain waste bins can be removed in preprocessing the graph.

On the other hand, node routing techniques can be applied in conditions where waste bins are sparsely located. In such situations, the waste bins will be represented as the nodes and the streets connecting the bins will be the edges.

2.3 Related works

The first paper about municipal garbage collection using vehicle routing techniques appeared in 1974 by Beltrami and Bodin [2], in which they attempted to tackle the problem of minimizing the total distance traveled over a planning period for customer (waste bins) with a set of feasible visit options. This is called the periodic vehicle routing problem (PVRP). This problem inspired a generation of researchers as a result of which, many papers have been written on the issue ever since.

A recent article (2012) [3] presented an adaptive large neighborhood search

algorithm to solve a waste collection vehicle routing problem with time windows, where the objective is to find cost optimal routes for garbage trucks such that garbage bins are emptied within a time window that takes into account the customer's available time and driver's working hours.

Another article [23] (2014) presents a complex waste collection problem that has fixed heterogeneous fleet of vehicles with different volume and weight capacities, fixed costs, unit distance running costs and hourly driver wage rates. It models the problem as a mixed binary linear program for which the researchers developed a local search heuristic with an optimality gap of 2% compared to the exact solution on small instances.

From Finland [25] (2005) described a guided variable neighborhood thresholding (GVNT) metaheuristic method for solving waste collection and transport problem in eastern Finland.

An ant colony algorithm (2012) for arc routing problem in waste collection is proposed in [18] to minimize the length of municipal solid waste collection routes. The solution takes into account one-way streets and forbids turns with narrow angles. The arc routing is transformed into node routing and a clustering-based multiple ant colony system algorithm is used.

Real-time information is used [14] (2012) in the optimization of recyclable waste collection, in particular glass waste, where garbage bins are fitted with fill-level sensors that provide real-time data to control center. The article proposes a collection policy based on three stages: first, calculation of routes after knowing the containers to collect in a day; second, estimate the containers to be collected in the coming days, and third, optimization of the routes based on the combination of the results in the first and second stages i.e. look for possible containers from those that are going to be emptied in the coming few days and add them to today's routes.

A simulation driven approach, in Kaohsiung Taiwan [4] (2015), is presented to support the optimization process. GIS is used for siting the drop-off stations and uses heuristic algorithm to solve the optimization problem. Also it compares the results of the heuristic algorithm with a genetic algorithm in the same GIS environment.

Meanwhile, a general overview of arc routing problems is given in [24]. In this collection of influential works by various researchers, different approaches to solving arc routing problems are presented. Some of these include, linear programming based methods, transformations and exact node routing solutions where the arc routing is transformed into node routing problem, matching theory to solve the Chinese postman problem etc. Also [5] (2010) presents recent results on arc routing problems.

2.4 Measuring algorithm performance

An algorithm is any well-defined computational procedure that takes some value as input and produces some value as output [6]. Computers perform tasks in steps. The number of steps taken by a computer to get something done are important indicators of an algorithm's performance. Assuming that all the computers in the world have equal computational power, i.e. memory and CPU, a function that takes in the input size of a problem and produces the number of steps needed to produce the output is

the running time of the algorithm. For instance, suppose we have a set of n numbers $S = \{1, 2 \cdots n\}$. The number of steps needed to sum n numbers in S is proportional to n. Therefore, we say summing numbers in S is a linear time operation. This means the algorithm is "summing numbers" and the running time is "linear".

Some algorithms exhibit different running times (functions) based on the nature and size of the input. These kinds of algorithms have often upper and lower bounds. An upper bound is a threshold for which an algorithm's running time never exceeds at any condition. And a lower bound is a threshold for which an algorithm will never complete a task before. And there is something in between which is often called *average case*. An upper bound and lower bound are referred to as *worst case* and *best case*, respectively.

We are often interested in the worst case and average case running times of an algorithm. The best case usually is mentioned for completeness.

2.4.1 Big-O Notation

Suppose some algorithm "A" takes $f(n) = 2n^2 + 3n + 4$ steps to do some task as a function of the input size n. We say the running time of "A" is $\mathcal{O}(n^2)$ — read as "Big-O of n-squared". That is because the lower order term n, the coefficients 2 and 3 and the constant 4 become irrelevant when the size of n is very large. We always take only the higher order term. To define precisely, assume two functions $f(n)$ and $g(n)$ are the running times of two algorithms on inputs of size n.

Definition 1 ([8]) *We say $f = \mathcal{O}(g)$ (meaning "f grows no faster than g") if there is a constant $c > 0$ such that $f(n) \leq c \times g(n)$*

Loosely, $f = \mathcal{O}(g)$ means $f \leq g$. Some of the most common types of running time functions are linear n, quadratic n^2, cubic n^3, logarithmic $\log(n)$, linearithmic $n\log(n)$, exponential 2^n etc. The growth rate of these running times as a function of input size is shown in Figure 3. It indicates that exponential functions grow faster than say, logarthmic functions. In algorithm, this means exponential functions take longer to perform the same task than linear or logarthmic for large n. In fact, exponential function algorithms have no use in practice. Factorial functions grow even faster than exponential functions and hence are practicaly useless to get a anything done.

As we can see from definition 1, the Big-O notation measures the worst case (upper bound) running time of an algorithm. Similarly, the lower bound is defined as $f = \Omega(g)$ (Big-Omega) meaning $g = \mathcal{O}(f)$. And in between we have $f = \Theta(g)$ (Big-Theta) to mean that $f = \Omega(g)$ and $f = \mathcal{O}(g)$ [8].

Generally, for any algorithm to be useful, its running time should be polynomial i.e. it has to be of the form n, n^2, n^3 etc. or less. In the next section, we see the behavior of algorithms that don't have polynomial running time.

2.4.2 Interactablity

A problem is regarded as tractable if and only if there is an algorithm for its solution whose running time is bounded by a polynomial in the size of the input [17]. The

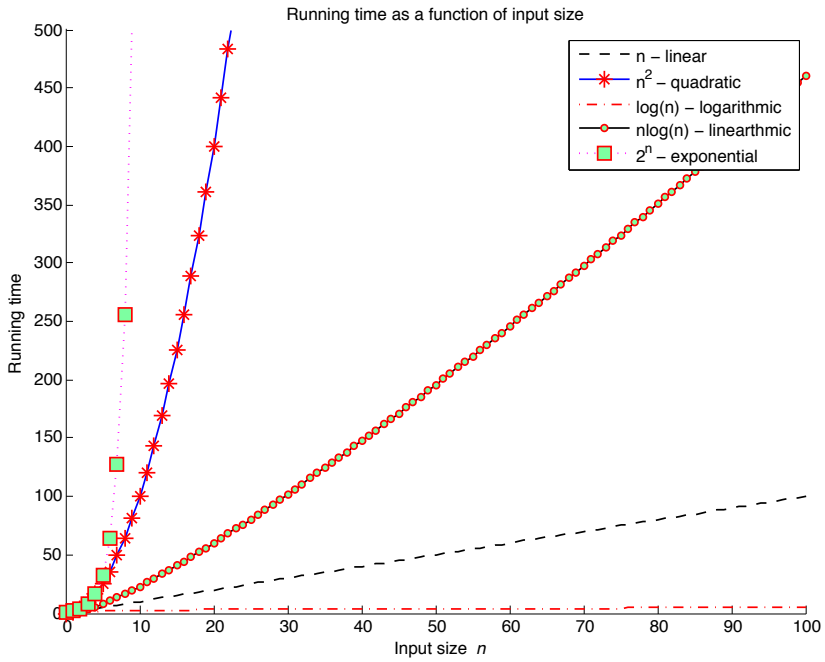


Figure 3: Running time function versus input size

class of all search problems that can be solved in polynomial time is denoted as P. Meanwhile, there are other class of search problems that have no known polynomial time algorithm. These class of problems are called NP short for "nondeterministic polynomial time" [8].

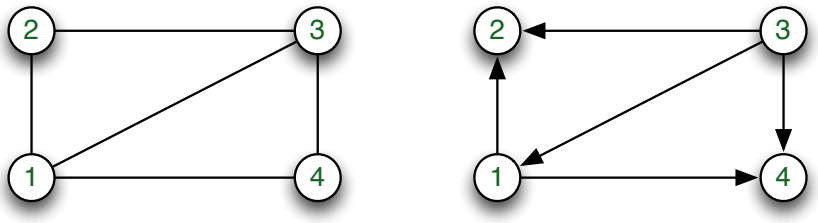
With this comes the notion of *NP-completeness*. A problem is called NP-complete, if other NP class problems can be polynomially transformed to it [15]. Such problems are of special kind, that if there exists a polynomial time solution to one, then there is polynomial time algorithm to all. It is worth noticing here that if a problem is NP-complete, it is highly unlikely there will ever be a polynomial time solution. Therefore, it a good reason to stop searching for a general solution to it. Some well known NP-complete[2] problems are the traveling salesman problem, integer linear programming, knapsack, capacitated arc-routing problem, rural chinese postman problem etc. The chinese postman problem in undirected and directed graphs can be solved in polynomial time as we will discuss in later sections.

[2]In complexity theory, NP-completeness is further classified into NP-easy and NP-hard. Throughout this paper it will all mean the same thing, i.e. the problem is hard.

2.5 Graph Theory

Graphs are mathematical structures that are used to model the pairwise relationship between entities. Typically, graphs are used to store network information where various algorithms operate on the data governed by the mathematical behaviour of the graph. A graph structure is made up of *vertexes* or *nodes* and *edges* or *arcs* that connects the nodes. The nodes usually represent entities such as points, cities, computers, people etc., whereas the edges represent the relations between the entities. For instance, distance between two cities, network wire between two computers etc.

A graph may be *undirected* where the relationship between the nodes has no directional meaning, for instance, friendship between two people; in *directed* or digraphs the relationship between the nodes has directional meaning, for instance, one way street between two points in a city, such relations are shown with a pointing arrow on a line, and is often called an *arc*. On the other hand, the term *edge* is often used in undirected graphs.



(a) Undirected graph with nodes and edges

(b) Directed graph with nodes and arcs

Figure 4: Graph representation

Formally, a graph G is defined as a pair of (V, E) or (V, A) for undirected and directed graphs respectively. Digraphs and undirected graphs often have equivalent mathematical properties. In the following sections, we will see some of these properties in some detail[3].

2.5.1 Basic definitions

Here is a list of basic concepts and definitions [9] in graph theory.

- *Order*: the order of a graph G is the number of vertexes it contains and is denoted as $|G|$. The order of the graph in Fig: 4a is 4.
- *Size*: the size of a graph G is the number of edges and is denoted as $\|G\|$. The size of the graph in Fig: 4a is 5.
- *Empty graph*: is a graph with no vertexes or edges and is denoted as $(\emptyset, \emptyset)$ or simply as $\emptyset$

[3]It is to be noted that Graph theory is a vast subject in math and computer science. The thesis covers only a small subset set of topics that will be necessary to understand the later sections.

- *Incident vertex*: a vertex v is incident with an edge e if $v \in e$.
- *Adjacency*: two vertexes x and y are adjacent if xy is an edge of G. Two edges p and q such that $p \neq q$ are adjacent if they have a common vertex. Adjacent vertexes or edges are also called *neighbors*.
- *Degree*: Also known as the *valency*; d of a vertex is the number of edges connected to v which is also equal to the number of *neighbor*s of v. A graph with all its vertexes having equal degrees is called a *regular* graph. Moreover, the minimum degree of a graph G is defined as $\delta(G) = \min\{d(v) : v \in V\}$. And the maximum degree of a graph is defined as $\Delta(G) = \max\{d(v) : v \in V\}$ which is the vertex with the maximum number of neighbors. A vertex with degree zero is intuitively called as isolated vertex. The average degree of a graph is the sum of all the degree of every vertex divided by the order of the graph, i.e. $|V|$ the number of vertexes in the graph.

$$d(G) = \frac{1}{|V|} \sum_{v \in V} d(v) = \frac{2|E|}{|V|} \tag{1}$$

Equation 1 naturally leads us to what is famously known as the *Handshaking Lemma*[4].

$$\sum_{v \in V} d(v) = 2|E| \tag{2}$$

Consequently, in every graph the number of vertexes of odd degree is even. This is a useful realization as we will find out later when we do graph augmentation to convert odd vertexes to make them even.

A vertex in a digraph has even degree if and only if the in-degree of the vertex is equal to the out-degree.The number of arcs going into a vertex v is the in-degree written $d^-(v)$ and analogously, the out-degree is written as $d^+(v)$. The imbalance or the difference between the in-degree and the out-degree of v is denoted by $\delta(v)$ i.e. $\delta(v) = d^+(v) - d^-(v)$. If $\delta(v) = 0$, then the vertex v is said to be balanced. The set of unbalanced vertexes with positive $\delta(v)$ is denoted as D^+ and the set of vertexes with negative $\delta(v)$ is denoted as D^- i.e.

$$D^+ = \{v : \delta(v) > 0\}$$
$$D^- = \{v : \delta(v) < 0\}$$

For example, in Figure 4b $D^+ = \{3, 1\}$ and $D^- = \{2, 4\}$.

- *Path*: A path is a sequential graph or element of a graph with vertexes and edges of the form, $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$ where x_0 and x_k are the end vertexes linked by P; and $x_1 \cdots x_{k-1}$ are the inner vertexes. The length of a path is determined by the number of edges and a path of length k is denoted as P^k.

[4]It is called the Handshaking Lemma because, the total number of people, each person at a party shakes hands with, will be twice the number of handshakes that occured.

– *Cycle*: is a cyclic sequential chaining of vertexes in a graph. Suppose, a path $P = \{x_0 \cdots x_{k-1}\}$ for $k \geq 3$, then a cycle can be formed by connecting the end vertexes as: $C := P + x_{k-1}x_0$. The length of a cycle is the number of edges or the number of its vertexes. Also, the minimum length of a cycle contained in a graph G is called the girth of G denoted as $g(G)$ and the maximum length of a cycle in G is called the circumference.

2.5.2 Euler trails and cycles

A *walk* $W = e_1e_2 \cdots e_n$ is a *trail*, if $e_i \neq e_j$ for all $i \neq j$. An Euler trail is one that visits every edge once. A connected graph G is *eulerian*, if it has a closed trail containing every edge of G. Such a trail is called an *Euler tour or cycle*.

Theorem 1 *A connected graph G is eulerian if and only if every vertex has an even degree.*

An earlier algorithm by Hierholzer(1873) finds an eulerian tour in an eulerian graph. It is probably the best available even today [24].

1. Start from an arbitary vertex v, gradually trace a cycle by following untraversed edges, until this procedure cannot be continued; in a connected graph, this happens only at v
2. If all edges have been traversed, stop.
3. Trace a second cycle starting from an unvisited edge incident to the cycle. Merge the two cycles into one. Go to step 2.

2.5.3 Connectivity

A graph $G = V \cup E$ is said to be connected if for any two vertexes $x, y \in V$ are linked by a path $P(x, y)$ in G. Connectivity is one of the very basic concepts in graph theory. Generally, it is concerned with determining the minimum number of elements that need to be removed to disconnect a connected graph. A maximal[5] connected subgraph X of G is called a *component* of G. A graph G is disconnected if and only if $G = G_1 \dot{\bigcup} G_2$ where G_1 and G_2 are non-empty subgraphs of G.

A vertex $v \in V(G)$ is called a cut-vertex if its removal makes a connected graph G disconnected. An edge $e \in E(G)$ is a bridge if and only if $V = V_1 \dot{\bigcup} V_2$ such that every path joining any $v_1 \in V_1$ to $v_2 \in V_2$ contains e. In other words, bridges in a graph are those edges that don't lie on any cycle in G. A rigorous verification of these claims can be found in a standard graph theory book.

[5] maximal with respect to set inclusion

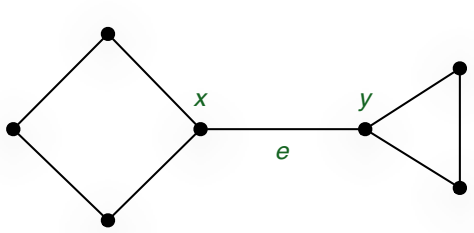


Figure 5: Vertex x and y are cut-vertexes and edge e is a bridge.

Also, if no two vertexes of G are separated by fewer than k other vertexes, then G is called *k-connected* for $k \in \mathbb{N}$. The greatest integer k such that G is *k-connected* is the *connectivity* $\kappa(G)$. Likewise for edges, if $|G| > 1$ and $G - F$ is connected for every set $F \subseteq E$ of fewer than l edges, then G is called *l-edge-connected*. The greatest integer l such that G is *l-edge-connected* is the *edge-connectivity* $\lambda(G)$. For a disconnected graph, $\lambda(G) = 0$. For every non-trivial (i.e. not isolated vertex) graph G, the following is true: [9]

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

The above inequality asserts that connectivity is always less than the minimum degree of a graph. But this doesn't mean that large minimum degree guarantees high connectivity. It only indicates the existence of highly connected subgraph.

In a digraph D, there are three types of connectedness [24]. These are:

- D is strongly connected if for all $x, y \in V(D)$ there exist paths $P(x, y)$ and $P(y, x)$.
- D is unilaterally connected if for all $x, y \in V(D)$ either $P(x, y)$ or $P(y, x)$ exists.
- D is weakly connected if its underlying graph G_D is connected i.e. if replacing all the directed edges with undirected edges results in a connected graph.

Clearly, a strongly connected digraph is unilaterally connected and unilaterally connected digraph is weakly connected but not vice versa. However, a weakly connected digraph is strongly connected if every arc belongs to a cycle.

Menger's Theorem

Menger's theorem states that for a graph $G = (V, E)$ and $A, B \subseteq V$, the minimum number of vertexes separating A from B in G is equal to the maximum number of disjoint (independent) paths from A to B in G. It is to be noted that, the set of vertexes in A is not adjacent with vertexes in B. This theorem is a fundamental realization in graph theory which is a foundation for many other concepts [24]. Yet again, the proof is left out for brevity but can be found in a standard graph theory book.

2.5.4 Matching

A bipartite graph is one whose vertex can be partitioned into two sets such that those in the same set have no connecting edges [24].

Theorem 2 *A graph G is bipartite if and only if G contains no cycles of odd length.*

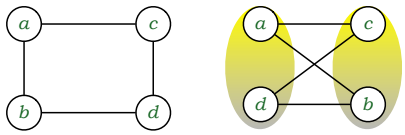
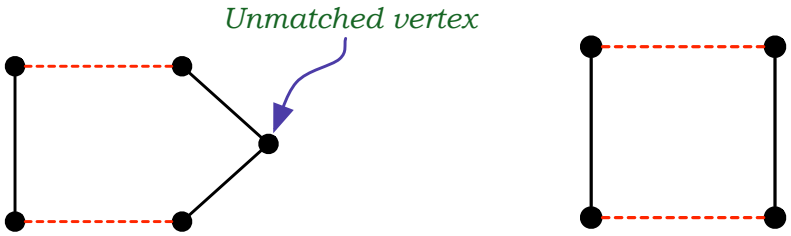


Figure 6: Bipartite graph

In a graph $G = (V, E)$, a matching M is a set of independent edges; i.e. no two edges share the same vertex. A vertex is *matched* if it has an end in the matching M; vertexes not incident with any edge of M are *unmatched* or *free*. A *perfect* matching is if all the vertexes in G are found in M [9].



(a) An unmatched vertex (b) A perfect matching

Figure 7: Matching in undirected graphs

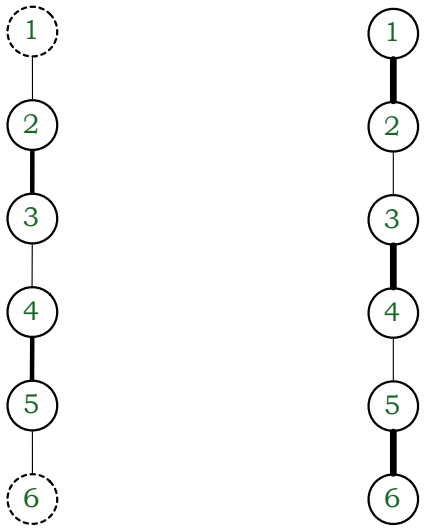
In a matching problem, the goal is often to find a matching containing as many edges as possible. This is called maximum-size matching. And the special case is to find a perfect matching or to verify that a perfect matching doesn't exist for G. In a weighted graph, where each edge has a weight, the goal is to find a matching of maximum total weight. There are important variants where the goal is to find a perfect matching of maximum or minimum total weight; among maximum size matchings, to find one of maximum or minimum total weight. There are four versions of matching:

- Unweighted bipartite
- Unweighted general
- Weighted bipartite a.k.a assignment problem
- Weighted general

Unweighted bipartite graph matching

In a fixed bipartite graph $G = (V, E)$ with bipartition $\{A, B\}$, how can we find a matching in G with as many edges as possible? For any matching M, a path that starts in partition A with unmatched vertex u and contains edges alternately from $E \setminus M$ and M, is called an *alternating path* [19]. In other words, an alternating path is one in which its edges are alternately in and out of the matching.

Now, an alternating path P that ends in unmatched vertex v of partition B is called an *augmenting path* i.e. an alternating path between two unmatched vertexes [19]. Naturally, this leads to the process of augmenting the matching i.e. given an augmenting path, change its unmatched edges to matched and vice-versa thereby increasing the size of the matching by one.



(a) Augmenting path (b) After augmentation

Figure 8: Maximizing matching by augmentation

The practical application of alternating paths is that, if we start with any matching and keep applying augmenting paths until no further such improvement is possible, the matching obtained will always be an optimal one, which is a matching with maximum number of edges [9]. Therefore, finding maximum matching amounts to finding augmenting paths iteratively. The algorithm for doing so is listed below.

Algorithm 1: Augmenting path algorithm for bipartite matching

Data: A bipartite graph G with partition $\{X, Y\}$
Result: A maximum cardinality matching M
Begin with an empty matching;
Direct all edges from X to Y;
while *there is unmatched vertex* x *in* X **do**
    search from x until reaching an unmatched vertex in Y or finish search;
    if *unmatched vertex found in* Y **then**
        augment;
        reverse direction of all arcs on the augmenting path;
    else
        delete all visited vertexes;
    end
end

Algorithm 1 applied on input graph Figure 9 gives result Figure 10.

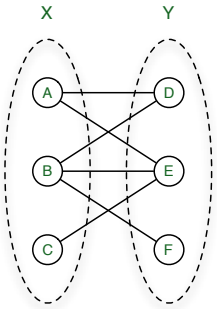
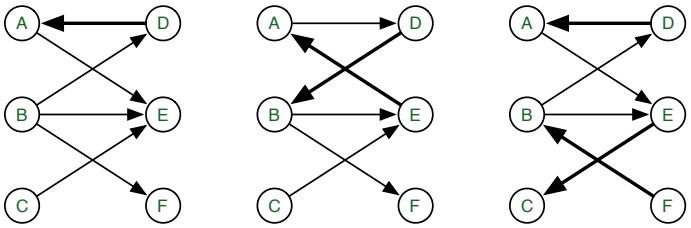


Figure 9: Input - Bipartite graph



(a) Search from A, find A, D and augment

(b) Search from B find B, D, A, E and augment

(c) Search from C, find C, E, A, D, B, F and augment

Figure 10: Iteratively augmenting to get maximum matching

The optimality of a matching can be checked based on the idea of *vertex cover*. A set $U \subseteq V$ is a *vertex cover* of E if every edge in the graph G is incident with

a vertex in U. König's theorem (1931) states that, the maximum cardinality of a matching in G is equal to the minimum cardinality of a vertex cover. This implies that the cardinality of an arbitrary matching M is always less than or equal to the minimum cardinality of a vertex cover but a maximum (optimal) matching is equal to the minimum cardinality of a vertex cover [9].

Weighted bipartite graphs

Algorithm 1 or some other equivalent algorithm are useful to solve matching in unweighted graphs. But how do we solve maximum cardinality matching when the edges are weighted? Such problem is very common in real life applications in many areas. In particular, we are interested in minimum weight matching in bipartite graphs.

Minimum weight matching is defined as: given a bipartite graph $G = (V, E)$ with bipartition (A, B) and weight function $w : E \to \mathbb{R}$ find a matching of minimum weight where the weight of matching M is given by $w(M) = \sum_{e \in M} w(e)$ [19]. An algorithm that can solve the minimum weight matching problem will be valuable to solve the optimization in the Chinese postman problem.

2.5.5 Graph data structures

So far, we have seen what a graph is and some of the basic concepts in graph theory. Now, we will see some data structures for dealing with graphs in an efficient way. A complete description of graph algorithm and data structures can be found on [6] or similar books. The material in this section is also based on [6]. There are two standard ways of representing graphs in computers. That is the *adjacency list* and *adjacency matrix*. In adjacency list, the vertexes are stored in a list structure where every element of the list is a pointer to another list of neighbors (adjacent vertexes) of the vertex at the current index.

Adjacency list is memory efficient when representing sparse graphs. However, it is not very efficient to check if two arbitrary vertexes are adjacent; in this case, adjacency matrix is better.

On the other hand, adjacency matrix stores graph data in a matrix form where every cell holds a value to indicate if two vertexes are connected by edge or not. Typically, in unweighted graph the cell value can be 0 or 1 if there is no edge connecting the vertexes or otherwise, respectively. In weighted graphs, the cell value can be the weight of the edge between the corresponding vertexes. The general rule according to [6] is that adjacency list is used on sparse graphs i.e. those for which $|E|$ is much less than $|V|^2$. For dense graphs where $|E|$ is close to $|V|^2$, adjacency matrix is recommended.

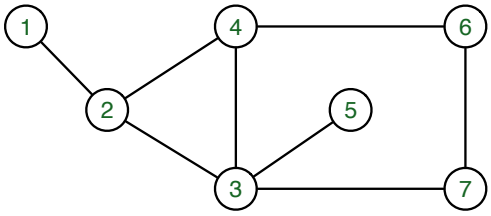
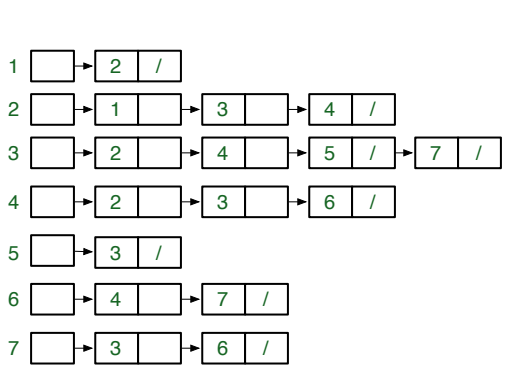


Figure 11: Example graph

The adjacency list and adjacency matrix representation of the graph on Figure 11 is shown in Figure 12



(a) Adjacency list

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	1	1	0	1
4	0	1	1	0	0	1	0
5	0	0	1	0	0	0	0
6	0	0	0	1	0	0	1
7	0	0	1	0	0	1	0

(b) Adjacency matrix

Figure 12: Graph representation

2.6 Chinese postman problem

A postman begins at the post office, traverses every street in his territory at least once and then returns to the post office. His objective, while delivering mails, is to walk as little as possible. This problem was first stated by Kwan Mei-Ko, a Chinese mathematician in 1962. It was later named *the Chinese postman problem* by Edmonds J.

Formally, let $G = (V, E \cup A)$ be a strongly connected graph where V is a set of vertexes, E is a set of undirected edges, and A is a set of directed arcs. Every edge $e = (v_i, v_j) \in E \cup A$ has a cost c_{ij} associated with it. A cost represents distance in our problem domain; however, it can be any arbitrary weight. There are several flavors of CPP depending on the nature of the edges or arcs [24]. These are:

The directed Chinese postman problem Unsurprisingly, the directed CPP is when the graph is made of only arcs i.e. $E = \varnothing$

The undirected Chinese postman problem The undirected CPP is when there are no arcs but only edges i.e. $A = \varnothing$

The mixed Chinese postman problem The mixed CPP is when the graph is made of both arcs and edges i.e. $A \neq \varnothing, E \neq \varnothing$

The windy Chinese postman problem The windy Chinese postman problem is where we've arcs with differing costs in either direction i.e. $A \neq \varnothing$ and $c_{ij} \neq c_{ji}$

The hierarchical Chinese postman problem The hierarchical Chinese postman problem is when a subgraph of G has precedence over another subgraph of G. For instance, if the postman has to service one part of city before the rest.

There are polynomial time algorithms for the Chinese postman problem on directed graphs and undirected graphs. However, the mixed CPP has been shown to be NP-complete [26]. Likewise, the windy and hierarchical CPP have both been shown to be NP-Hard [24].

In this thesis, we are particularly interested in the directed CPP because the directed graph model resembles the real world road network better than the undirected and it's still solvable in polynomial time, of course with some limitations which we will discuss later.

3 Results

So far, we have studied the basic properties of graphs which serves as the building block to the solution of many combinatorial problems. In the following sections, the algorithm for solving the directed chinese postman problem is presented, the implementation details are discussed and test results are shown.

3.1 Solving the DCPP

A strongly connected digraph $D(V, A)$ is Eulerian if and only if every vertex has even degree. That means in digraphs the in-degree and out-degree of every vertex v should be equal, or $\{\delta(v) = 0 : \forall v \in V\}$. Thus, if a graph is Eulerian then an Eulerian cycle can be found, which is an optimal solution to the DCPP. However, when a graph is not Eulerian the solution to the DCPP is no longer straightforward. This section describes an algorithm for the DCPP based on Thimbelby [33].

The basic idea revolves around augmenting the odd nodes to become even so that eventually we will have an Eulerian graph for which we can find an Eulerian tour, hence, the Chinese postman tour. Of course, augmenting odd nodes means to figure out which edges to repeat while keeping the overall cost at minimum. Repeating edges means walking without delivering mails. This problem can be tackled in various ways.

One possible method is the transportation algorithm as shown in [24]. Let I be the set of vertexes v_i for which the number of incoming arcs exceeds the number of outgoing arcs by s_i and J be the set of vertexes v_j for which the number of outgoing arcs exceeds the number of incoming arcs by d_j. The reader can interpret s and d as supply and demand, respectively. Then, we can drive the *transportation problem* as follows:

$$
\begin{aligned}
&\text{Minimize} \qquad \sum_{v_i \in I} \sum_{v_j \in J} c_{ij} x_{ij} \\
&\text{subject to} \\
&\sum_{v_j \in J} x_{ij} = s_i, \qquad (v_i \in I) \\
&\sum_{v_i \in I} x_{ij} = d_j, \qquad (v_j \in J) \\
&x_{ij} \geq 0
\end{aligned}
\tag{3}
$$

Of course, we are interested in the variable x_{ij}, which is the number of repetitions of the arc (v_i, v_j), i.e. the number of times the postman has to traverse the arc (v_i, v_j) keeping the overall distance minimum. This is a classic linear programming approach.

Consider the following graph on Figure 13

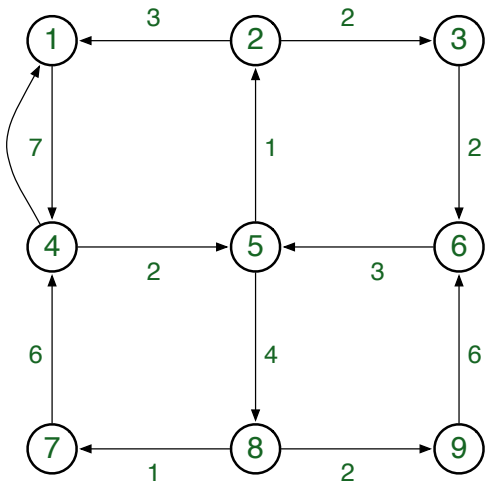


Figure 13: Input digraph

Observe that it's a strongly connected, i.e. there is a path from each node to every other node, and vertexes $\{1, 6, 2, 8\}$ are odd vertexes since there is an imbalance between the incoming and outgoing arcs. The handshaking lemma we saw previously, tells us that, for undirected graphs there can only be an even number of odd vertexes. For a digraph, the handshaking lemma is slightly modified; the sum of in-degree vertexes is equal to the sum of out-degree vertexes. In this particular case, half of the odd vertexes belong to the set I and half belong to J.

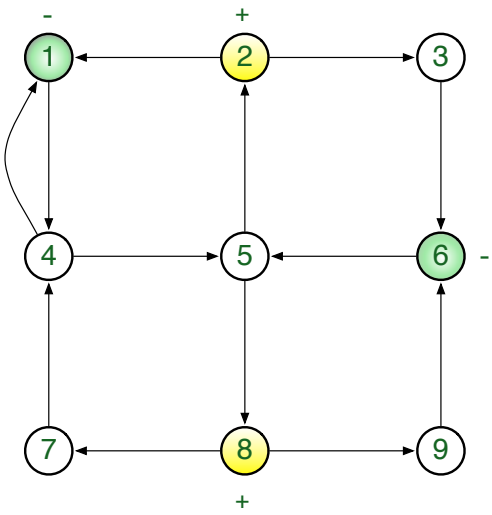


Figure 14: Odd vertexes classified as +ve and -ve nodes

Let's put the -ve and +ve vertexes into I and J bipartition so we can apply Equation 3 on it.

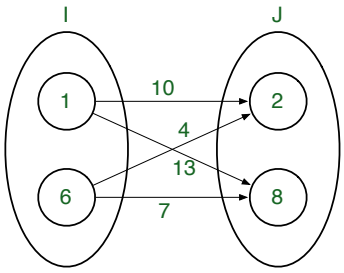


Figure 15: Bipartite graph of the odd vertexes with shortest distance weights between the vertexes

For convenience we use the variable names as $x_{11} = v_1v_2$, $x_{12} = v_1v_8$, $x_{21} = v_6v_2$, $x_{22} = v_6v_8$.

$$\begin{aligned}
&\text{Minimize } (c_{11}x_{11} + c_{12}x_{12}) + (c_{21}x_{21} + c_{22}x_{22}) \\
&\qquad = (10x_{11} + 13x_{12}) + (4x_{21} + 7x_{22}) \\
&\text{and the constraints} \\
&\qquad x_{11} + x_{12} = 1 \\
&\qquad x_{21} + x_{22} = 1 \\
&\qquad \text{and} \\
&\qquad x_{11} + x_{21} = 1 \\
&\qquad x_{12} + x_{22} = 1 \\
&\qquad x_{ij} \geq 0
\end{aligned} \tag{4}$$

Solving this system of linear equation yields $x_{11} = 1$, $x_{12} = 0$, $x_{21} = 0$, and $x_{22} = 1$. Or alternatively, $x_{11} = 0$, $x_{12} = 1$, $x_{21} = 1$, and $x_{22} = 0$ is an optimal solution. And the resulting augmented digraph is shown in Figure 16.

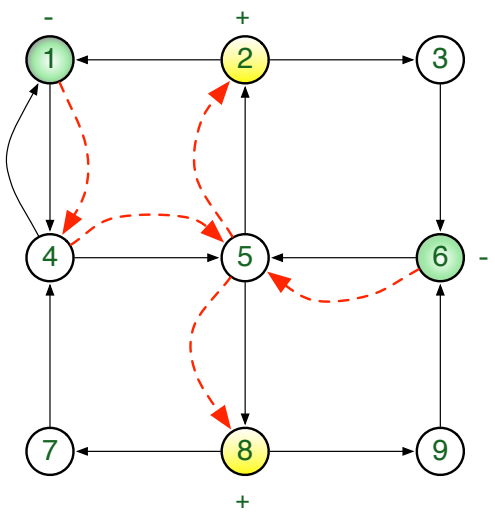


Figure 16: Eulerian graph of the input graph

Finally, the last step in obtaining a Chinese postman tour is to compute the Eulerian tour of the augmented graph. It is to be noted that there are alternative methods to do the augmentation process. One such method is the minimum weighted bipartite matching a.k.a the assignment problem. Generally speaking, linear programming methods are similar in essence. The minimum weighted bipartite matching can be solved using the *Hungarian algorithm* [33], however, it is not be a simple algorithm to implement.

The flowchart on Figure 17 shows the broader picture of the steps involved.

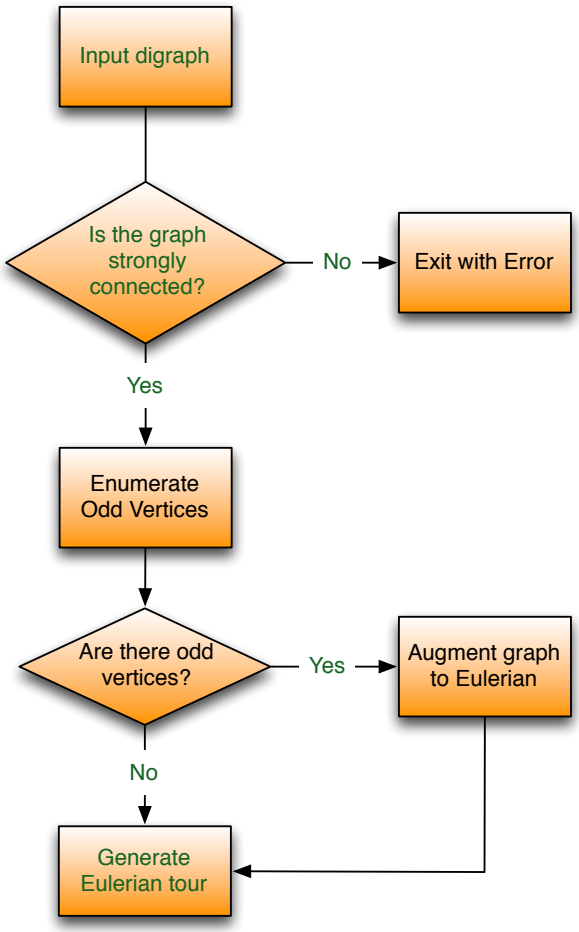


Figure 17: Flowchart for steps involved in solving the CPP

3.2 Implementation

The implementation of the DCPP is based on a previous implementation by Thimbelby [33]. The code is written in Node.js® compatible javascript and demonstrates the algorithm on small size weighted multidigraphs. Multidigraphs are digraphs with possibly repeated arcs between vertexes.

The implementation uses adjacency-matrix for representing the graph data. It has two main constructors, namely `Adapter` and `CPP`. The CPP object is the core of the algorithm responsible for producing the expected output whereas the Adapter is responsible for handling input data and transforms it into a form that is efficient for the CPP to deal with. The CPP object has an instance variable called `arcs` which is a 2-dimensional array that stores the arcs between the nodes. It also has instance variable `delta` which is a 1-dimensional array and corresponds with the variables s_i and d_j, in the previous sections, which are the imbalances at vertex v_1 and v_2. Every time a new arc (v_1, v_2) is added, we increment `arcs[v1][v2]` and `delta[v1]` and

decrement `delta[v2]`.

3.2.1 Shortest paths

The CPP object also keeps a 2-dim array called `path` which maintains the shortest path between every pair of vertexes. Essentially during initialization, `path[v1][v2] = v2`. However, this value will get updated when low cost paths are discovered later in the program.

All-pairs shortest path algorithm is used in this case. In particular, We use the Floyd-Warshall algorithm, which is an efficient all-pairs shortest path algorithm and runs in $\Theta(V^3)$ [6]. Johnson's algorithm on sparse graphs also runs faster but on dense graphs the performance is the same to that of Floyd-Warshall's algorithm. Also it may be more complicated to implement.

Some key observations of the algorithm are:

- A shortest path doesn't contain the same vertex more than once
- For a shortest path from i to j such that any intermediate vertexes on the path are chosen from the set $\{1, 2, \cdots, k\}$, there are two posibilities:
  - k is not a vertex on the path, so the shortest such path has length d_{ij}^{k-1}
  - k is a vertex on the path, so the shortest such path is $d_{ik}^{k-1} + d_{kj}^{k-1}$
- Define $d_{ij}^{(k)}$ to be the weight of the shortest path from vertex i to j for which all intermediate vertexes are in the set $1, 2, \cdots, k$. Consequently, $d_{ij}^{(k)}$ can be defined recursively as follows

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 0 \end{cases}$$

where w_{ij} is the weight of an arc in the adjacency matrix of the digraph $D(V, A)$ and has the following values:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{cost of arc } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in A \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin A \end{cases}$$

The Floyd-Warshall algorithm typically only computes the cost of the paths between all pairs of vertexes. But we can take advantage of it to build the predecessor matrix denoted as Π. This will allow us to compute not only the cost but also the paths. That is done by computing a sequence of matrices $\Pi^{(0)}$, $\Pi^{(1)}$,... $\Pi^{(n)}$ at the same time while computing $D^{1,2...k}$.

Let's define $\pi_{ij}^{(k)}$ as the predecessor of vertex j on a shortest path from vertex i with all intermediate vertexes in the set $\{1, 2, \ldots, k\}$. There is a recursive formulation [6] of $\pi_{ij}^{(k)}$ where:

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ or } w_{ij} < \infty \end{cases}$$

which is the case when $k = 0$ i.e. the shortest path from i to j has no intermediate vertexes. For $k \geq 1$ we have,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

The Pseudocode for the Floyd-Warshall's algorithm is shown on listing 2.

Algorithm 2: Floyd-Warshall All-pairs shortest path algorithm [6]

Input: W Adjacency matrix of $D(V, A)$
Output: $D^{(n)}$ least cost matrix
n = vertexes
D = W
Π initialization
for $k = 1$ *to* n **do**
  for $i = 1$ *to* n **do**
    for $j = 1$ *to* n **do**
      if $d_{ij} > d_{ik} + d_{kj}$ **then**
        $d_{ij} = d_{ik} + d_{kj}$
        $\pi_{ij} = \pi_{kj}$
      end
    end
  end
end
return $D^{(n)}$

Furthermore, Algorithm 2 can be modified to check for negative cycles while computing least cost paths. This can be achieved by adding an if statement in the inner loop [33].

$$\begin{cases} \text{stop! -ve cycle found} & \text{if } i = j \text{ and } w_{ij} < 0 \\ \text{proceed} & \text{otherwise} \end{cases}$$

3.2.2 Inspecting strongly connectedness

Strongly connectedness is a necessary condition to solve the DCPP and therefore should be checked in the beginning stages of the computation. This can be achieved by using the Floyd-Warshall's algorithm predecessor matrix $\Pi^{(n)}$. In the matrix, if any of the $w_{ij}^{(n)}$ is NIL for $i \neq j$, then the graph is not strongly connected. Unsurprisingly, when the graph is not strongly connected, we exit with error message.

3.2.3 Searching for odd vertexes

We keep track of the delta, which is the difference between the number of incoming and outgoing arcs, of each vertex during the insertion of arcs. The indices of the delta array represents the vertexes. Thus, we iterate through the delta and copy the negative and positive vertexes into two different arrays `neg` and `pos`. Therefore, we now have the odd vertexes in two classes i.e. `neg` and `pos` arrays and we have the corresponding deficiencies in the `delta` 1-dim array.

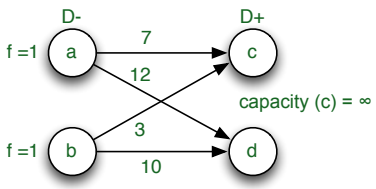
3.2.4 Optimization

The algorithm used to solve the optimization problem is called cycle-cancelling. It is a well-known combinatorial method to solve minimum cost flow problems [13]. There are other methods to solve the optimization problem, namely linear programming, matching [11] etc.; however, they are complicated to implement. In particular, applying linear programming to solve Equation 3 should be done carefully. Because the results, while being feasible, may not be integral [33]. This means that the output x could be in $\mathbb{R}$ when it should be in $\mathbb{Z}$.

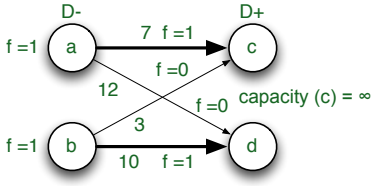
Cycle cancelling is based on the negative cycle optimality condition. The negative cycle optimality condition says that, for a feasible flow to be optimal, there should be no negative cycle in the residual graph. It starts with a feasible solution i.e., one that converts all odd vertexes to even but the augmentation is not optimal, and iteratively improves by removing negative cycles from the residual graph for the feasible solution. In minimum cost flow problems, the initial feasible max flow can be computed using Edmonds-Karp algorithm [22].

In the implementation, a greedy algorithm from [33] is used. It works, by iteratively combining odd vertexes in D^-, those with -ve imbalance, with those with +ve imbalance in the set of D^+.

The application of algorithm 3 on input Figure 18a might produce a result as in Figure 18b which is not optimal but is feasible.



(a) Input to algorithm 3



(b) One of many (two in this case) feasible solutions

Figure 18: Finding initial feasible solution

Unsurprisingly, the worst case running time of algorithm 3 is $\mathcal{O}(|V|^2)$ in which case every vertex on the graph has odd degree.

Once we have a feasible solution, it is the input to the cycle cancelling algorithm. The general cycle-cancelling algorithm for minimum cost flow is shown in listing 4

Algorithm 3: Greedy algorithm to find feasible solution [33]

Input: Set of -ve and +ve vertexes $\in D(V, A)$
Output: Feasible solution
n = no of vertexes in D^-
m = no of vertexes in D^+
for $i = 1$ *to* n **do**
    u = neg[i]
    for $j = 1$ *to* m **do**
        v = pos[j]
        if *-delta[u] < delta[v]* **then**
            $x_{uv} = -delta[u]$
        else
            $x_{uv} = delta[v]$
        end
        $delta[u] = delta[u] + x_{uv}$
        $delta[v] = delta[u] - x_{uv}$
    end
end
return x

and can be adapted to our problem.

Algorithm 4: Cycle cancelling algorithm min cost flow problem [22]

Input: Directed network $D(V, A)$
Output: Minimum cost optimal solution
Establish a feasible flow x in in the network
Let $A_1 = \{(i, j) \in A : f((i, j)) < c((i, j))\}$
Let $A_2 = \{(i, j) \in A : f((j, i)) > 0\}$
Let $D_x = (V, A_1 \cup A_2)$be the residual network
while D_x *contains a negative cycle* **do**
    Identify a negative cycle W
    $d := min\{r_{ij}|(i, j) \in W\}$
    `/* augment` d `units of flow along` W `and update` D_x `*/`

    $x((i, j)) = x((i, j)) + d, \forall (i, j) \in A : (i, j) \in A_1 \cap W$
    $x((i, j)) = x((i, j)) - d, \forall (i, j) \in A : (j, i) \in A_2 \cap W$
end
recover an optimal flow x from the final residual network D_x

In Algorithm 4, c is the capacity of the arc, which in our case can be assumed to be ∞, f is the flow; in our case, f is equal to the δ_v. The cost will be the same as before which is the distance.

Now let's define the residual network $D_f = (V, A_f)$ associated with flow f.

Suppose $e = (u, v) \in A$ then:

- if $f_{uv} < c_{uv}$ then $(u, v) \in A_f$ with cost ω_{uv} and residual capacity $r_{uv} = c_{uv} - f_{uv}$
- if $f_{uv} > 0$ then $(v, u) \in A_f$ with cost $-\omega_{uv}$ and residual capacity $r_{vu} = f_{uv}$

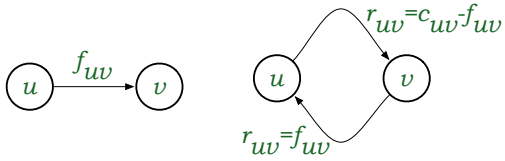


Figure 19: A flow network D (left) and its residual network D_f (right)

Let's apply the cycle-cancelling algorithm on the feasible solution in Figure 18b we found earlier. The residual graph of the network is shown in Figure 20.

1. First search for a negative cycle. The Floyd-Warshall algorithm already does that for us. The sum of costs on the negative cycle is $-7 + 12 + 3 + -10 = -2$. Therefore, we have a negative cycle shown with dotted lines on Figure 20.

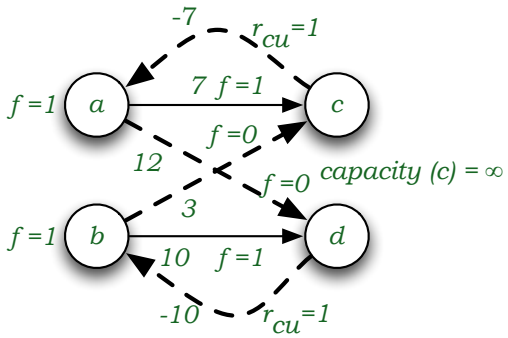


Figure 20: The residual network of the feasible solution in Figure 18b

2. Now, we have to destroy the negative cycle. To do that, we need to find the minimum residual capacity on the cycle. The residual capacity $r_{ac} = 1, r_{db} = 1, r_{ad} = \infty,$ and$r_{bc} = \infty$. Therefore, $d = min\{1, 1, \infty, \infty\} = 1$

3. Next we do the following two steps:

$$f((i,j)) = f((i,j)) + d,\ \forall (i,j) \in A : (i,j) \in A_1 \cap W$$
$$f((i,j)) = f((i,j)) - d,\ \forall (i,j) \in A : (j,i) \in A_2 \cap W$$

where W is the negative cycle. Thus, flow f along ac and bd becomes zero i.e. $1 - d = 0$ and flow along, bc and ad becomes $0 + d = 1$.

4. At this point, we have another feasible solution. Calculate the residual network of the new solution and see if it has any negative cycles. If it has negative cycle we repeat the steps above again and the iteration goes on. If it has no negative cycle then that solution is optimal.

Assuming that all the data of the problem are integral, the algorithm terminates within $\mathcal{O}(|E|CU)$ iterations [1] where C is maximum capacity and U is maximum cost of the network. Taking into account the Floyd-Warshall algorithm for finding the negative cycles the global cost of the algorithm amounts to $\mathcal{O}(|V|^3|E|CU)$. Alternatively, one can use the Bellman-Ford algorithm for finding negative-cycles with a running time of $\mathcal{O}(|E||V|)$ [6]. In addition, during the execution of the Floyd-Warshall algorithm, exponentially large numbers may occur which will likely cause an overflow for larger graphs. To avoid this problem, the if statement for checking negative cycles in the inner loop, can be put outside the nested loops and iterate over the diagonals of the distance matrix; without affecting the worst-case running time [16].

3.2.5 Route generation

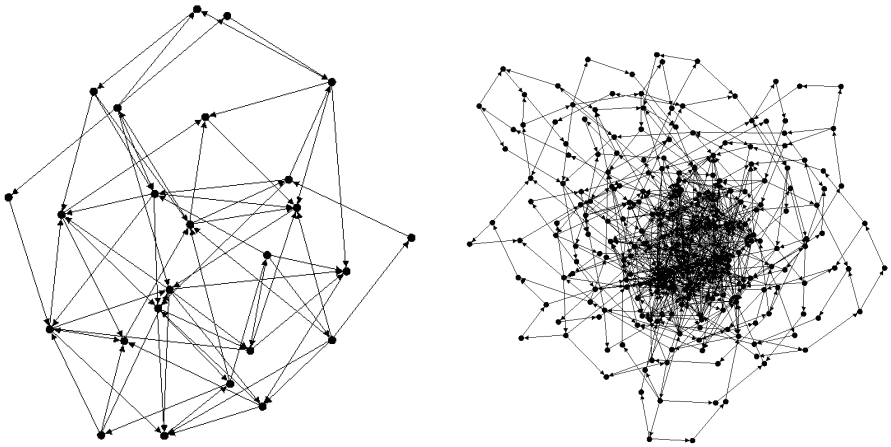
At this point, the optimization problem is solved and it's possible to construct the Eulerian cycle which is the Chinese postman tour. The algorithm for the Eulerian cycle uses a spanning tree, which we have as a by-product from the Floyd-Warshall algorithm predecessor matrix. It works by iterating over all the vertexes i and following the shortest path to another vertex j if there is one; and decrements the visited edges from the global array as each path is followed. If there is no path $i \rightsquigarrow j$, take another arc outside the spanning tree. Otherwise, take the last arc from the spanning tree to the start vertex [33].

3.2.6 Testing

The test data is a strongly connected, Gnuttella peer-to-peer file sharing network, digraph from the Standford large network dataset collection [30]. The nodes represent hosts in the network topology and edges represent connections between the hosts. The purpose of the test is to probe the performance of the algorithm (Such a test would have been better performed on a road network; however, at the time of writing, a digraph road network was unavailable). The original dataset has 6301 nodes and 20777 edges in total; and 2068 nodes and 9313 edges in the strongly connected component. The edges were iteratively removed to reduce the size of the graph and a weight of unit value is assigned to each arc, since the original data has no weights.

Two external Java (JGraphT and Graphstream) libraries were used for visualization, and pre-processing of the data.

The asymptotic running time in the previous section reveals that the running time of the algorithm is pseudopolynomial, i.e. it is not quite polynomial and it behaves more like exponential time algorithm. The test was conducted on a Macbook Pro with a duo core processor of $2.4GHz$ and memory of $8GB$ on a single thread of execution.



(a) 25 nodes and 68 edges (b) 224 nodes and 740 edges

Figure 21: Test data — peer-to-peer file sharing (Weight$(v_i, v_j) = 1$ *unit*)

Table 1: Test results

Nodes	**Edges**	**Tour length**	**Cost**	**Phi**	**Time (sec)**
5	8	16	16	8	0.006
9	13	18	63	17	0.006
25	68	135	135	67	0.053
224	740	2779	2779	2039	5.348
858	3273	10234	10234	6961	22185.414 (6.2 hrs)

Incidentally, the result of the algorithm for the graph on Figure 13 is shown below.

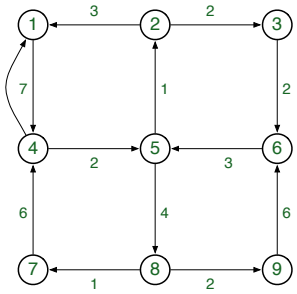


Figure 22: Input graph (Same as Figure 13)

Resulting route = (1) → (4) → (5) → (8) → (9) → (6) → (5) → (2) → (3) → (6) → (5) → (8) → (7) → (4) → (1) → (4) → (5) → (2) → (1)
Phi = 17 and Cost = 63

4 Discussion

At this point, we have figured out an algorithm to solve an unconstrained directed Chinese postman problem. This algorithm can be used in a mobile map, web map or packaged as a plugin to be used with any GIS software suites. In fact, at the moment there is an experimental CPP solver plugin [28] for an open source software QGIS.

In the context of GIS, suppose a scenario where a city has to estimate the travel cost of grit or salt spreading during the winter season. Such problem is a direct application of the Chinese postman problem. And an algorithm like the one we developed is an important tool in tackling the problem. Waste collection, which is the theme of this thesis, is also an important example of where the CPP can be applied and used.

As we have noted in Section 2.3, most of the methods employed in solving the garbage collection problem are indeed esoteric, non-optimal and very complicated to implement. In comparison, the DCPP solution we have is easy to implement and understand. Of course, it has its downsides as well. The DCPP cannot be used in all instances of garbage collection route optimization. In particular, in cases where garbage collection is done selectively,i.e. when there is no need to traverse the entire road network, then the result of the algorithm clearly would be non-optimal. Thus, it shouldn't be seen as a silver bullet that can solve all waste collection routing problems.

As a further matter, it is important to note that this is unconstrained solution. The capacity of waste trucks is assumed non-finite, garbage can be collected at anytime, there are no narrow angle turning restrictions, as long as the direction of the arc is not violated, and there are no other trucks doing the same job in the given network. As such, this is not a single solution package in and of itself but certainly, part of the solution. In times, where we have strong constraints, we can use other methods that are suited to each individual constraints. Yet when there are no constraints or the constraints are loose, we can employ the DCPP algorithm to do the job.

The general approach for using the DCPP algorithm in GIS context either for routing waste collection trucks, estimating fuel costs during salt and grit spreading etc. is as follows:

1. **Determine the service area:** The first thing that needs to be done is to determine the underlying road network of the service area. This can be achieved by clipping the network based on some given bounding box. In doing so, it is necessary to make sure that the underlying network remains connected. The process of clipping can be automated using clip software tools.

2. **Identify intersection:** Road intersections are what make up the vertexes in the adjacency matrix of the graph. The longitude and latitude information is not necessary for our algorithm since it is purely combinatorial and not geometrical as that is natural to graphs. However, the latitude and longitude coordinates can be used later, once the route computation is finished, to plot

the route on map. The intersections should be labeled, that can be done using reverse geocoding.

3. **Assign distance:** The distance between the intersections should be assigned and should be the same in both directions if the streets are bi-directional. Multi-lane streets can be shown as parallel edges.

4. **Run the algorithm:** The result is a list of street names in the order required by the DCPP.

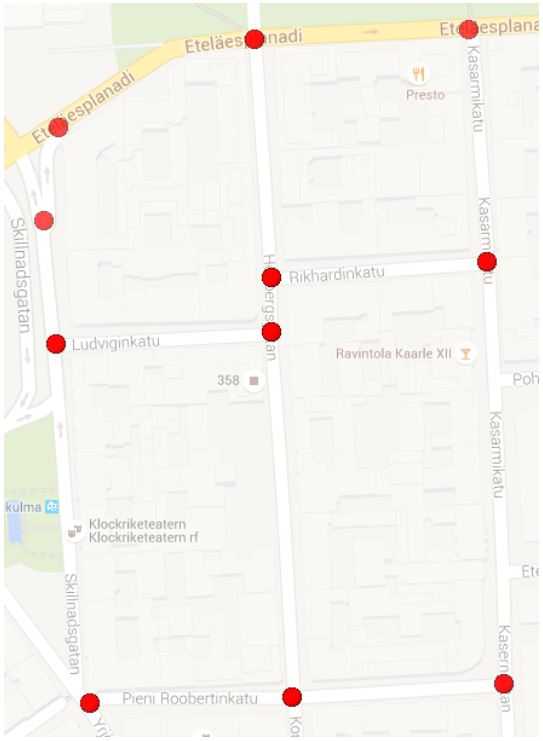


Figure 23: Example service area — The points indicate street intersections. Garbage bins are located along the streets between the intersections. The truck will have to drive along all streets to empty the bins.

Table 2: Street data in the service area given as a directed graph. This network information can be input in csv format.

From	Y	X	To	Y	X	Distance(m)
Ludvigsgatan 5-7	24.943972	60.165764	Erottajankatu 19	24.943863	60.166307	62.38
Erottajankatu 19	24.943863	60.166307	Eteläesplanadi 24	24.943998	60.166716	49.13
Eteläesplanadi 24	24.943998	60.166716	Högbergsgatan 36	24.94574	60.167109	109.57
Högbergsgatan 36	24.94574	60.167109	Högbergsgatan 43	24.945884	60.166055	118.62
Högbergsgatan 43	24.945884	60.166055	Högbergsgatan 36	24.94574	60.167109	118.62
Högbergsgatan 43	24.945884	60.166055	Högbergsgatan 30	24.94588	60.165814	27.01
Högbergsgatan 30	24.94588	60.165814	Högbergsgatan 43	24.945884	60.166055	27.01
Ludvigsgatan 5-7	24.943972	60.165764	Högbergsgatan 30	24.94588	60.165814	106.09
Lilla Robertsgatan 11-13	24.944271	60.164178	Ludvigsgatan 5-7	24.943972	60.165764	178.59
Lilla Robertsgatan 7-9	24.946062	60.164202	Lilla Robertsgatan 11-13	24.944271	60.164178	101.35
Lilla Robertsgatan 11-13	24.944271	60.164178	Lilla Robertsgatan 7-9	24.946062	60.164202	101.35
Högbergsgatan 30	24.94588	60.165814	Lilla Robertsgatan 7-9	24.946062	60.164202	179.5
Lilla Robertsgatan 7-9	24.946062	60.164202	Högbergsgatan 30	24.94588	60.165814	179.5
Lilla Robertsgatan 7-9	24.946062	60.164202	Lilla Robertsgatan 1	24.947944	60.16426	104.25
Lilla Robertsgatan 1	24.947944	60.16426	Lilla Robertsgatan 7-9	24.946062	60.164202	104.25
Lilla Robertsgatan 1	24.947944	60.16426	Kaserngatan 44	24.947793	60.166127	208.82
Kaserngatan 44	24.947793	60.166127	Lilla Robertsgatan 1	24.947944	60.16426	208.82
Kaserngatan 44	24.947793	60.166127	Högbergsgatan 43	24.945884	60.166055	106.93
Högbergsgatan 43	24.945884	60.166055	Kaserngatan 44	24.947793	60.166127	106.93
Eteläesplanadi 14	24.947631	60.167152	Kaserngatan 44	24.947793	60.166127	115.41
Högbergsgatan 36	24.94574	60.167109	Eteläesplanadi 14	24.947631	60.167152	107.01

The addresses in the CSV data were obtained by using reverse geocoding on google maps. The result of this algorithm is shown below starting and ending at *Ludvigsgatan 5-7*:

Result — shortest tour

Ludvigsgatan 5-7 → *Erottajankatu 19*
Erottajankatu 19 → *Eteläesplanadi 24*
Eteläesplanadi 24 → *Högbergsgatan 36*
Högbergsgatan 36 → *Eteläesplanadi 14*
Eteläesplanadi 14 → *Kaserngatan 44*
Kaserngatan 44 → *Högbergsgatan 43*
Högbergsgatan 43 → *Högbergsgatan 30*
Högbergsgatan 30 → *Lilla Robertsgatan 7-9*
Lilla Robertsgatan 7-9 → *Lilla Robertsgatan 11-13*
Lilla Robertsgatan 11-13 → *Lilla Robertsgatan 7-9*
Lilla Robertsgatan 7-9 → *Högbergsgatan 30*
Högbergsgatan 30 → *Lilla Robertsgatan 7-9*
Lilla Robertsgatan 7-9 → *Lilla Robertsgatan 11-13*
Lilla Robertsgatan 11-13 → *Ludvigsgatan 5-7*
Ludvigsgatan 5-7 → *Högbergsgatan 30*
Högbergsgatan 30 → *Högbergsgatan 43*
Högbergsgatan 43 → *Högbergsgatan 36*
Högbergsgatan 36 → *Högbergsgatan 43*
Högbergsgatan 43 → *Kaserngatan 44*
Kaserngatan 44 → *Lilla Robertsgatan 1*

```
Lilla Robertsgatan 1      →    Kaserngatan 44
Kaserngatan 44            →    Högbergsgatan 43
Högbergsgatan 43          →    Högbergsgatan 30
Högbergsgatan 30          →    Lilla Robertsgatan 7-9
Lilla Robertsgatan 7-9    →    Lilla Robertsgatan 1
Lilla Robertsgatan 1      →    Lilla Robertsgatan 7-9
Lilla Robertsgatan 7-9    →    Lilla Robertsgatan 11-13
Lilla Robertsgatan 11-13  →    Ludvigsgatan 5-7
Phi is 871
Cost = 3283
```

As we can see in the output, the shortest possible tour for the service area takes about 3.3km. `Phi` is the distance of the repeated streets. This is the optimal tour. In other words, there is no other better way to walk along the streets in Figure 23 than the one given here.

5 Conclusion

This thesis has attempted to describe all the necessary algorithmic machinery needed to solve the directed Chinese postman problem. The DCPP is a very good model to simulate real life route optimization in household waste collection, snow ploughing, street cleaning [10] and other applications, where minimal arc traversing is the objective, without being hopelessly intractable. Yet, it is also necessary to note that there are many variables in real life that will elevate the difficulty of the problem of routing for household waste collection.

In particular, in situations where it is not necessary to visit all edges but a small subset of them, the DCPP is not very helpful. The Rural postman problem is a variant of the Chinese postman problem where the objective is to visit only a small subset of the arcs. Unfortunately, the RPP is NP-hard [29]. In addition, in the case where the postman has capacity and time window constraints the problem is called capacitated arc routing problem with time windows. Clearly, this is also an NP-hard problem since it's evolved from the RPP. While there are many heuristic and exact methods [20, 24, 29, 34] for these classes of problems, they are very complicated for practical use.

Meanwhile, the algorithm presented in this thesis comparatively is simple to understand and implement. This version of cycle-canceling doesn't specify the order for selecting negative cycles. A variant of this algorithm with a judicious choice of cycles for canceling has a strongly polynomial time [13]. The network simplex algorithm, which is believed to be one of the fastest for minimum cost flow problems, is a type of cycle-canceling algorithm [1] and as a result it is one of the widely used algorithm to solve minimum cost flow problems. Furthermore, scaling cycle-canceling algorithm that works by identifying negative cycles with "sufficiently large" residual capacity and augmenting flows along them, is also a strongly polynomial algorithm [31].

From our observation of the DCPP so far, it can be used effectively in densely populated networks where waste bins are located along every street. An open tour DCPP can also be computed with the same technique for cases where the vehicle enters the network at one point and exits at another. Capacity and time-window constraints perhaps can be handled in a scheduler separately from the routing. The results from the scheduler can be used as input to a network generator that gives out the underlying road network. On top of the road network, we can compute the DCPP algorithm like the one presented in this thesis. Such arrangement is being investigated at the moment and is a potential future research subject.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] E. J. Beltrami and L. D. Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94, 1974.

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

[4] Ni-Bin Chang and Ana Pires. *Optimal Planning for Solid Waste Collection, Recycling, and Vehicle Routing*, pages 515–551. John Wiley & Sons, Inc., 2015.

[5] Angel Corberán and Christian Prins. Recent results on arc routing problems: An annotated bibliography. *Networks*, 56(1):50–69, 2010.

[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[8] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill Education, May 2006.

[9] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[10] Benjamin Dussault. Phd dissertation: Modeling and solving arc routing problems in street sweeping and snow plowing, 2012. Dissertation directed by: Bruce Golden.

[11] Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman, 1973.

[12] http://global.britannica.com/topic/Konigsberg-bridge-problem Encyclopedia Britannica: Online. Königsberg bridge problem. Accessed on 2015-11-09 16:05.

[13] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, October 1989.

[14] R. Grosso de la Vega, J. Muñuzuri Sanz, M. Rodriguez Palero, and J. Teba Fernandez. Optimization of recyclable waste collection using real-time information. In J. Carlos Prado-Prado and Jesús García-Arca, editors, *Annals of Industrial Engineering 2012*, pages 171–177. Springer London, 2014.

[15] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[16] Stefan Hougardy. The floyd–warshall algorithm on graphs with negative cycles. *Inf. Process. Lett.*, 110(8-9):279–281, April 2010.

[17] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[18] Jie Liu and Yanfeng He. Ant colony algorithm for waste collection vehicle arc routing problem with turn constraints. In *Computational Intelligence and Security (CIS), 2012 Eighth International Conference on*, pages 35–39, Nov 2012.

[19] L. Lovász and M.D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. American Mathematical Soc., 2009.

[20] C. Martinez, I. Loiseau, M.G.C. Resende, and S. Rodriguez. Brkga algorithm for the capacitated arc routing problem. *Electronic Notes in Theoretical Computer Science*, 281:69 – 83, 2011. Proceedings of the 2011 Latin American Conference in Informatics (CLEI).

[21] Christoph Manuel Meyer. *Vehicle Routing under Consideration of Driving and Working Hours*. Gabler, 2011.

[22] García Sanjuan Fernando; Martínez Villaronga Adrià A.; Puigcerver Pérez Joan; Melzer Michaela. Description of cycle canceling algorithm for solving the minimum cost flow problem. *Estructures Matemàtiques per a la Informàtica II*, 2009.

[23] Markov Ilya; Varone Sacha; Bierlaire Michel. Vehicle routing for a complex waste collection problem, 2014. Published in: Proceedings of the 14th Swiss Transport Research Conference, Monte Verità - Ascona, May 14-16 2014, 2014, p.1-27.

[24] Dror Moshe. *Arc Routing, Theory Solutions and Applications*. Springer, 2000.

[25] Teemu Nuortio, Jari Kytöjoki, Harri Niska, and Olli Bräysy. Improved route planning and scheduling of waste collection and transport. *Expert Syst. Appl.*, 30(2):223–232, 2006.

[26] Christos H. Papadimitriou. On the complexity of edge traversing. *J. ACM*, 23(3):544–554, July 1976.

[27] Sari Piippo. *Municipal Solid Waste Management in Finland*. Thule Institute, 2013.

[28] http://plugins.qgis.org/plugins/chinesepostman/version/0.1/ Ralf Kistner: Online. Chinese postman solver 0.1 experimental, May 2013. Accessed on 2015-11-09 16:05.

[29] Ana M. Rodrigues and José S. Ferreira. Cutting path as a Rural Postman Problem: solutions by Memetic Algorithms. *International Journal of Combinatorial Optimization Problems and Informatics*, 3(1):31–46, 2012.

[30] https://snap.stanford.edu/data/p2p-Gnutella08.html SNAP Stanford Large Network Dataset Collection: Online. Gnutella peer-to-peer network, August 2002. Accessed on 2015-11-09 16:05.

[31] P. T. Sokkalingam, Ravindra K. Ahuja, and James B. Orlin. New polynomial-time cycle-canceling algorithms for minimum cost flows. *NETWORKS*, 36:53–63, 1996.

[32] http://stat.fi/til/jate/2012/jate_2012_2014-05-15_tie_001_en.html Statistics Finland: Online. Sixty-six per cent of combustible waste is burned, May 2014. Accessed on 2015-11-09 16:05.

[33] Harold Thimbleby. The directed chinese postman problem. *In journal of Software – Practice and Experience*, 33:2003, 2003.

[34] R. van Bevern, C. Komusiewicz, and M. Sorge. Approximation algorithms for mixed, windy, and capacitated arc routing problems. *ArXiv e-prints*, June 2015.