T.C.
ISTANBUL AYDIN UNIVERSITY
INSTITUTE OF SOCIAL SCIENCES

VEHICLE ROUTING PROBLEM SECTORIZATION USING HYBRID METHODS

MSc. THESIS

Mohammed Aref MANSOUR

Department of Business
Business Administration Program

Thesis Advisor: Assist.Prof.Dr. Uğur ŞENER

March 2018

T.C.
ISTANBUL AYDIN UNIVERSITY
INSTITUTE OF SOCIAL SCIENCES

VEHICLE ROUTING PROBLEM SECTORIZATION USING HYBRID METHODS

MSc. THESIS

Mohammed Aref MANSOUR
(Y1412.130030)

Department of Business
Business Administration Program

Thesis Advisor: Assist.Prof.Dr. Uğur ŞENER

March 2018

T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ MÜDÜRLÜĞÜ

Yüksek Lisans Tez Onay Belgesi

Enstitümüz İşletme İngilizce Anabilim Dalı İşletme Yönetimi İngilizce Yüksek Lisans Programı **Y1412.130030** numaralı öğrencisi **M. Aref MANSOUR**'un **"VEHICLE ROUTING PROBLEM - SECTORIZATION USING HYBRID METHODS"** adlı tez çalışması Enstitümüz Yönetim Kurulunun 05.03.2018 tarih ve 2018/08 sayılı kararıyla oluşturulan jüri tarafından oybirliği ile Tezli Yüksek Lisans tezi olarak kabul edilmiştir.

Öğretim Üyesi Adı Soyadı	**İmzası**
Tez Savunma Tarihi :29/03/2018	
1)Tez Danışmanı: Dr. Öğr. Üyesi Uğur ŞENER	.....
2) Jüri Üyesi : Dr. Öğr. Üyesi Günay Deniz DURSUN	.....
3) Jüri Üyesi : Dr. Öğr. Üyesi Ali KABLAN	.....

Not: Öğrencinin Tez savunmasında **Başarılı** olması halinde bu form **imzalanacaktır**. Aksi halde geçersizdir.

ACKNOWLEDGMENTS

First of all I would like to thank my family for their support throughout my academic life, without you guys I would not have been able to get to this point. Thank you!

I wish there were a way to thank my friend, teacher, and advisor Mr. Fırat Bayır in person, unfortunately though, I can only pray that your soul rests in peace and that my work has lived up to your expectations. Mekanın cennette olsun.

A huge thank you to my supervisor Mr. Uğur Şener who was patient enough to help me in my writing within the last 2 years.

My best friend in Portugal Mrs. Anna Maria Rodrigues, although I try my best but I know I could not thank you enough for your support and help while I was your guest, so please accept my sincere appreciation.

I would like to thank my friend Mr. Bruno Oliveira for writing the SIMPLEX part of the algorithm and helping me with my research.

Most importantly, all the thanks to Almighty Allah for helping me finish this work. May it be a blessed step and an addition to human knowledge, alhamdolillah!

.

Mohammed Aref MANSOUR

FOREWORD

This thesis is written in completion of Master's Program in Business Administration at Istanbul Aydin University. At first the thesis studies the history of the problem since its introduction in the 1950s by defining the problem, discussing its effect on our daily lives and its variations. Afterwards, the thesis studies the main approaches in which the problem has been approached since its introduction. The three approaches which are (1) Exact methods, (2) Heuristics, and (3) Metaheuristics are then studied in details along with their different variations and implementations as well as their pros and cons.
The thesis then describes the new algorithm that has been developed to address the issue using a hybrid of metaheuristics and exact methods to take advantage of both methods and overcome their shortcomings. Afterwards, the algorithm is tested using real world data with two sets of configurations and the test runs are compared to one another. The algorithm is then benchmarked against (VRP Spreadsheet Solver) to test its asses how the algorithm performs when compared to other utilities and tools available in the market today.

March 2018 **Mohammed Aref MANSOUR**

DECLARATION

I hereby declare that all information in this thesis document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that as required by these rules and conduct, I have fully cited and referenced all material and results, which are not original of this thesis (29/03/2018)

Mohammed Aref MANSOUR

TABLE OF CONTENTS

ABBREVIATIONS

ACO	: Ant Colony Optimization
ATSP	: Asymmetric Traveling Salesman Problem
EM	: Exact Methods
GA	: Genetic Algorithms
GRASP	: Greedy Randomized Adaptive Search Procedure
SA	: Simulated Annealing
SC	: String Cross
SE	: String Exchange
SM	: String Mix
SP	: Set Partitioning
SR	: String Relocation
TS	: Tabu Search
TSP	: Traveling Salesperson
VF	: Vehicle Flow
VND	: Variable Neighborhood Descent
VNS	: Variable Neighborhood Search
VRP	: Vehicle Routing Problem

LIST OF FIGURES

LIST OF TABLES

ARAÇ ROTALAMA PROBLEMI
HİBRİT YÖNTEMLERİ İLE SEKTÖRIZASYON

ÖZET

1959 yılında ilk olarak ortaya atılan Araç Rotalama Problemi, geçen yıllarda önem kazanmaya devam etmiştir. Buna karşılık bu problemin en uygun sonucunu tam olarak bulan bir algoritma henuz geliştirilmemistir. Bu araştırmamızda ilk Araç Rotlama Problemi'nin tanıtımına yer verilmiştir. Bununla Araç Rotalama Problemi'nin iş dünyasındaki önemi, bu problemin sozumesnin. Gözükesinin sağlayacağı faydalar ve problemin sınırlamalarından bahsedilmektedir.
O tezde, geçmişte bu konuyla ilgili olarak yapılan tüm araştırmalar hakkında bilgilere verilmiş ve tiplerine göre ayrıştırılmıştır. Sonrasında ise en uygun sonucu bulan kesin ve bir amaca yönelik sezgisel yöntemleri kullanan bir algoritma tarafımızdan oluşturmuştk. Bu algoritma, sezgisel yöntemler kullanılarak müşteriler küçük gruplara ayırmaktadır. Her bir grupta olan müşteri, bir araca ait olup, sonrasında yapılan kesin metotlar ile bütün müşterilerden, her aracın depodan çıkış bölümünden geçip, tekrar depoya dönme yolu bulunmaktadır. Bu bölümde müşteri adedi az olduğundan dolayı kesin metotlar kullanılmaktadır. Bu nedenle bulunan yol planının, en iyi plan olduğu kesindir. Sonrasında algoritmanın bulduğu sonuç, geçmişte bulunan sonuçlar ile karşılaştırılmış ve ona göre ya (1) aynı sonuç ile yoluna devam eder, ya (2) yolu değişir, ya da (3) şu ana kadar en iyi bulunan sonucu vererek tamamlanır. Bu tezi, söz konusu algoritmayı test etmek için, merkez Ankara seçilmiş olup, Ankara'dan başlayarak sekiz (8) araç ile Türkiye'nin diğer kalan seksen (80) ilinden geçerek, yeniden Ankara'ya dönmesi için en az masraflı olan yol planı bulunmaya çalışılmıştır. Belirtilen bu bilgiler iki (2) kez test edilmiş olup, her defasında farklı ayarlar kullanılmıştır. Sonrasında ise aynı ayarlar ile 2017 yılında Prof. Güneş Erdoğan'nın tarafından Microsoft Ofis Programı olan Excel kullanılarak geliştirmiş olan uygulama ile performansları karşılaştırılmıştır. Performans çıktı sonuçlarının analizi yapılıp, her aracın avantajları, dezavantajları, faydaları, kusurları ve bu kusurların sebepleri anlatılarak açıklanmaya çalışılmıştır. Yapılan bu testler, Microsoft Excel kullanılan bu uygulama ile aynı ayarla, aynı zamanda ve çok daha iyi ve başarılı sonuçlar verdiğini göstermiştir.

Anahtar Kelimeler: *VRP, Araç Rotlama Problemi, Exact Methodları, Metaheuristics, Heuristics, Lojistik, Algoritma.*

VEHICLE ROUTING PROBLEM
SECTORIZATION USING HYBRID METHODS

ABSTRACT

VRP (Vehicle Routing Problem) is a problem that was first introduced in the late 1950s and has been since studied thoroughly. However there are no algorithms that have the ability to conclude an optimum solution for the problem yet.
In this paper a brief introduction is given to familiarize the reader with the problem. Afterwards scope of the study along with its limitations, and assumptions are expressed briefly. Motivations are explained as well to indicate the importance of the problem and its effects in our everyday lives. A summary of previously done researches along with their types of solutions are studied throughout this paper.
Then a new algorithm that uses a combination of exact methods and metaheuristics to find the solution closest to the optimum one is introduced. The algorithm in question uses metaheuristics to divide the clients population into smaller populations called sections where each section represents a group of clients that will be served by one of the available vehicles. The algorithm then finds the best route within each of the sections using exact methods which have the advantage of guaranteeing best solutions for small numbers of clients within acceptable time windows. The algorithm then compares newly found solutions with previous ones and decides accordingly whether it must (1) continue in the same path, (2) change it, or (3) stop processing and outputs the best solution that has been found until this moment as the best solution possible.
Afterwards, the algorithm is tested using two different sets of configurations to find the best way to visit all 80 cities in the Republic of Turkey with 8 vehicles starting and ending at Ankara with the lowest possible cost. The algorithm is then benchmarked against a tool that has been developed by Dr. Erdoğan Güneş and makes use of Microsoft Excel to find the optimal solution for the same problem with the exact same configurations and circumstances and then both tools are compared to one another stating advantages of each of them. The comparison shows Dr. Güneş's excel tool was more successful in finding the better solution within the same time window allowed for processing the given data. At the end of the paper, applications of the algorithm, along with suggestions for further improvements are suggested.

Keywords: *VRP, Vehicle Routing Problem, Exact Methods, Metaheuristics, Heuristics, Logistics, Delivery, Pick-up, Supply chain.*

1. INTRODUCTION

This chapter helps better introduce the problem via its 6 sections, which help divide the introduction into its respective categories as follows; Scope of Study, Problem Definition, Limits, Assumptions, Hypothesis, and Motivation respectively.

1.1 Scope of Study

This thesis studies the history of VRP (Vehicle Routing Problem), its impact in our lives, and approaches which were taken to solve it along with a new algorithm that combines metaheuristics and exact methods to solve it.

1.2 Problem Definition

The problem can be defined as the problem of finding the closest solution to optimum solution for a VRP, following constraints bound the problem:

- There is one depot from which all trips will originate and end after making a full trip. The depot vertex will be the vertex with index 0, V_0
- There is a definite number of vehicles $m > 0$
- There is a definite number of vertices (clients) to be visited which is at least equal to the number of vehicles $n \geq m$
- All vertices (clients) should be visited exactly once, by exactly one vehicle.
- The runtime time window is defined prior to execution start.
- The coordinates for the vertices are provided.

- The costs of trips between vertices are provided with regards to at least one cost factor.
- There is an importance factor associated for each cost factor and is provided beforehand.
- The sum of importance factors must be equal to one $\sum_{i=0}^{CN} CF_i = 1$

1.3 Limits

Theoretically speaking the size of data is unlimited, although it should be taken into account that a single vehicle has a maximum visits number (tour capacity) of a 100 vertices (clients) since it is the maximum limit for the TSP solving algorithm being used.
The algorithm is limited to one depot, and works in accordance with the constraint that all vehicles will make trips that originate and terminate at the depot.

1.4 Assumptions

Since the algorithm was designed with logistic companies in mind, it is assumed that it will have a time window in which it will be able to find the best possible solution for the problem provided, it is also assumed that the time available would be known beforehand.
Also because of the same design purpose, it is assumed that the data and locations would be provided before hand and would not change during runtime.
The Algorithm takes into consideration whatever cost data fed to it and makes the best use out of it.

1.5 Hypothesis

The method in question is supposed to take advantage of exact methods' precision but will overcome their performance issues by depending heavily on metaheuristics to divide the clients' population into smaller groups groups that can be run using exact methods within acceptable run times.
Using a combination of both exact methods and metaheuristics means that the algorithm will be able to perform with performance close to metaheuristics' without

suffering from the precision drawback associated with metaheuristics because it uses exact methods as well to keeps precision in check. The hypothesis can be summed as follows:

- **H_1:** Our algorithm is faster than exact methods
- **H_2:** Our Algorithm is more accurate than Metaheuristics.

1.6 Motivation

Logistics takes part in almost every single aspect of human lives, from the delivery of final products to the consumer to transportations of raw and half-ready materials necessary for production. With such importance in our lives and being such a vital component of the production cycle we can only expect it to have as much effect on our lives.

Logistics problems do not stop at being small problems since they are becoming bigger and more vital day by day. Our constantly changing lives mean that our cities are becoming bigger, number of packages being sent and received is getting bigger, and the demand for faster more reliable logistics (next day delivery, or even same day delivery are now available in some parts of the world) is always on the rise.

Here are 6 of the most important issues that each logistics company faces on daily bases and would benefit a lot from the improvement or solutions introduced to the problem:

1.6.1 Greenhouse gas emissions (CO_2 footprint)

Although it is hard to estimate the greenhouse gas emissions of the transportation and logistics sector, it has been estimated that freight transportation alone produced 33.7M tons of CO_2 in the year 2004 in the UK alone, which accounts for 6% of the total CO_2 emissions in the UK in the same year. (Mckinnon, 2007).

1.6.2 Big amounts of goods, big numbers of clients

At the year 2016 it was announced that the total trade volume in the world was 16.2 Trillion U.S Dollars (World Trade Organization, 2016), this number is

expected to rise to a whopping 27 Trillion U.S Dollars (Reuter, 2010). Moreover at the year 2015 there were 7.3 Billion people on the face of earth, but this number is projected to grow to 8.5 Billion people by the year 2030 and continue to rise until it reaches 9.725 Billion people by the year 2050 (United Nations, 2015).
Looking at these number we can understand the importance that the logistics sector will hold in the near future.

1.6.3 Struggle with human resources

In India at the year 2011 it was estimated by The National Skill Development Organization of India that 7.3 Million people were employed by the logistics sector, but that number is going to rise to a 25 Million by the year 22, this means that logistic firms will have to find 17 Million qualified employees in just 11 years. suffices to say there will be a huge pressure on the sector.
A similar example can be found in the United States where 400 Thousands truck drivers were needed at 2010 alone, and because of retiring professional drivers in the United States the market should supply itself with 1 Million professional truck drivers between the years 2012, and 2027.
Not only land transportation is facing such problems as according to The Ocean Policy Research Foundation up to 364,000 seafarers shortfall is estimated by 2050. Airlines will also suffer from 460,000 pilots and 650,000 technicians and maintenance employees shortage as well (PwC, 2015).

1.6.4 Time importance

According to Consumer Intelligence Research Partners Amazon has an estimate of 80 Million Prime subscribers in the United States alone(Dunn, 2017) which is double what Amazon has expected only two years before, Each of these subscribers spends double the normal spending on Amazon annually (Dunn, 2017). Prime, which is a service that among many of its promises promises delivery of goods purchased on Amazon within one day (Amazon, 2017).
All of these figures mean there is a growing huge number of clients expecting their products to be delivered in almost real-time on daily bases all around the world, thus the demand for flexible fast and reliable logistics will increase as well.

1.6.5 Planning and unexpected scenarios

When the airport of Bangkok was forced to close because of the protests in the year 2008, the Thai economy suffered a loss of 8.5 Billion US Dollars, not only the tourism and airline businesses were hit by the closure, but exports were hit as well, an example of that would be the orchid export business which contributes to 80% of the orchid export business in the world and suffered a 9 Million US Dollars as a consequence of the closure.

Another incident of an unexpected scenarios is the eruption of a volcano in Iceland which happened in 2010 that forced airlines to cancel 100,000 flights in the course of 6 days causing a loss of 1.6 US Dollars for the airlines industry which was felt most by airliners that operate in Europe (Ruske et al., 2011).

1.6.6 Big resources in use

Given all the reasons and studies given before in section 1.6, it can be easily assumed that the logistics sector is among the most resources-demanding sectors, and given that it comes to direct contact with most if not all of the industries it can be only assumed that with time it is going to get more pressing, important, and demanding unless new game-changing technologies are introduced into the business. We are starting to see some of these technologies surfacing at the time being but not on a large scale or for big amounts of goods, but rather for the delivery of package to the final consumer. An example of new technologies being used is the usage of drones by online shopping firms such as Amazon Inc. to deliver purchases to clients, but such technologies are yet too far from introducing a real change in the industry.

Looking at the figures and statistics mentioned above in section 1.6, it can be concluded that the role of logistics in the global market is only going to get bigger and more important, so it makes sense to constantly work on making the sector better and more efficient.

2. LITERATURE REVIEW

In this chapter some resources and previous studies in the field are looked at according to the field (sectorization, and routing)

2.1 Sectorization

Sectorization (or districting) problems consist of partitioning a large region into smaller sub-regions (sectors), to facilitate the management of some activities (Mourão et al., 2009).

2.1.1 Graphs

A *graph* G consists of a finite nonempty set V of p points together with a set X of q unordered pairs of distinct points of V. Each pair $X = \{u, v\}$ in X is a *line* of G, and if $\{u, v\} \in X$, then u and v are said ot be *adjacent* in G. Moreover, if $X = \{u, v\} \in X$, then x is said to be *incident* with the points u and v. (Alba, 1973).

Graphs have many usages in demonstrating real-life problems like demonstrating roads and conjunctions between the roads, while showing some important information about the roads and the vertices between them. other usages might be in chemical problems to demonstrate the chemical compounds and the way they being produced.

2.1.1.1 Graphs and multigraphs

The difference between graphs and multigraphs is that multigraphs are basically graphs that have at least two nodes that are connected to each others using more than one edge. However multigraphs cannot have loops in them (loops are nodes that are connected to themselves using an edge).

The set $V(G)$ is called the vertex set where $E(G)$ is called edge set of the multigraph G.

The number of vertices in a multigraph $v(G)$ is referred to as the order of the multigraph, while the number of edges in a multigraph $e(G)$ is referred to as the size of the multigraph.

2.1.1.2 Matrices and multigraphs

As multigraphs consist of sets of vertices and nodes we can represent them using matrices which is very helpful in finding solutions for problems using computers as matrices are easy to deal with in computers.

It helps as well that in graphs the size of dots and the shapes of the edges do not demonstrate real-life characteristics of our problem.

To demonstrate a multigraph using a matrix, a square matrix should be defined with each adjacent representing the number of edges connecting two vertices to each other given that the column of the adjacent represents one vertex and the row of it represents another vertex (it is the same vertex when it is a diagram adjacent).

2.1.1.3 Vertex degree

Two vertices u and v in a G are said to be **adjacent** if they are joined by an edge, say, e in G. In the case when e is the only edge joining u and v, we also write $e = uv$, and we say that: (1) u is a **neighbour** of v and vice versa. (2) the edge e is **incident with** the vertex u (and v). (3) u and v are the two **ends** of e. (Koh et al., 2007).

Given a vertex v in G, the **degree** of v in G, denoted by $d_G(v)$, is the defined as the number of edges incident with v.(Koh et al., 2007).

A vertex v is called an **isolated-vertex** if $d(v) = 0$; it is called an **end-vertex** if $d(v) = 1$.(Koh et al., 2007).

An $end-vertex$ in a multigraph G is a vertex v with a degree $d(v) = 1$. Although an $end-vertex$ has nothing to do with the end of an edge.(Koh et al., 2007).

In general the *sum of vertices degree is double the* ***multigraph degree***. (Koh et al., 2007).

A vertex v in a graph G can be either **odd** or **even** and that would be determined by its **degree** $d(v)$.(Koh et al., 2007).
Each multigraph G has a **maximum degree** $\Delta(G)$ and a **minimum degree** $\delta(G)$ which are defined by the maximum degree of vertices and the minimum degree of vertices in the multigraph G (Koh et al., 2007).

2.1.1.4 Regular graphs

A graph G is referred to as a **regular graph** if the degree of all the **vetrices** in the graph are of the same degree. If a graph G is **regular** and all of its **vertices** are of the degree K it is referred to as K**-regular graph** (Koh et al., 2007).
It should be noted that given a **regular graph** G, the **maximum degree**, and the **minimum degree** of G will be equal and both will be equal to K, thus $\Delta(G) = \delta(G) = K$. There are three special types of multigraphs:

2.1.1.4.1 Null graphs A graph G is called a **null graph (empty graph)** if $E(G)$ is **empty**, or $K = 0$. Which translates to a graph that has no **edges** at all. A null graph of order n is denoted by N_n.(Koh et al., 2007).

2.1.1.4.2 Complete graph A graph G is called a **complete graph** if every vertex is a neighbor for all the other vertices in the graph. It is denoted by K_n.(Koh et al., 2007).
Complete graphs are **always** $(n-1) - regulargraphs.$

2.1.1.4.3 Cycles A graph G, where $V(G) = \{v_1, v_2, v_3, ..., v_n\}$ is called a **cycle** if any given vertex v_i is an **adjacent** to v_{i-1} and vv_{i+1}. A **cycle** of the order n is referred to as C_n and is called an $n - cycle$.(Koh et al., 2007).

2.1.1.5 Paths and walks

A **walk** in a multigraph G is an alternating sequence of vertices and edges that begins and ends at vertices and is referred to as (#) and would be written as:

$$(\#)\, v_0 e_0 v_1 e_1 v_2 ... v_{k-1} e_{k-1} v_k$$

where $k \geq 1$ and e_i is **incident** with v_i and v_{i+1} for each $i = 0, 1, ..., k-1$. The walk (#) is also called a $v_0 - v_k$, v_0 is called the **initial vertex**, while v_k is called the **terminal vertex**. The length of the walk (#) is defined as k which is the number of edge occurrences in the sequence.(Koh et al., 2007).

A walk is called a **trail** if no *edge* in it is traveressed more than once.

A trail is called a **path** if no *vertex* in it is visited more than once.

A walk is called a **closed** walk if it starts and ends at the same node, such as $initialvertex = terminalvertex$.

A closed walk of length at least two with no repeated edges is called a **circuited**.

A circuit is called a **cycle** if no vertex is repeated (except the initial and terminal vertices).

We call the *shortestpath* between two vertices a and b the **distance** between them, and referred to as $d(a, b)$.

In a graph G the **diameter** is defined by the *greatest distance* between any two vertices in the graph.

2.1.1.6 Connectedness of a graph

A graph can consist of one or more **components** where each component is a set of vertices connected to each other (using edges).

A *component* can consist of any number of nodes $k \geq 1$.

A **connected** graph is a graph that consists of *one component*, thus a walk can be done between *any* two vertices in the graph.

A graph is called **disconnected** if it is not a *connected* one. In other words, if it consists of *more than one component*.(Koh et al., 2007).

2.2 Routing - VRP

VRP (Vehicle Routing Problem) is a well-known problem that was first formally introduced in the year 1959 (Laporte, 2009).

The Vehicle Routing Problem (VRP) can be defined as a problem of finding the optimal routes of delivery or collection from one or several depots to a number of cities or customers, while satisfying some constraints (Yeun et al., 2008). Because of the different scenarios that are being faced in practice it might be more useful

to consider VRP as a class of problems rather than a problem. But no matter what we consider it this problem (class of problems) is being faced on daily basis at delivery posts and logistics companies, and is a central problem for load distribution departments at each firm. This might give an indication to the reason that it has been studied for such a long time and captured as much attention.

The *classical VRP* would be defined as follows. Let $G(V, A)$ be a directed graph where $V = \{0, ..., n\}$ is the vertex set and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the arc set. Vertex 0 represents the depot while the rest of the vertices in the set represent the clients (locations that should be visited or make a delivery to). A fleet of m identical vehicles of load limit Q is based in the depot. Each customer has a specified demand (load) q_i. A cost matrix c_{ij} represents the cost for travel between vertices in the set V. The VRP can solved such as designing m vehicle routes that start and end at the depot while fulfilling the goal of operation which is visiting each client once and only once by exactly one vehicle, the solution is subject to two constraints: (1) the total demand of a route cannot exceed Q, and (2) the total length of the route cannot exceed a preset limit L (Laporte, 2009).

In principle the VRP generalizes the well-known *Travelling Salesman Problem* (TSP) but with the generalization process it made the problem so much more complex for there exists exact algorithms capable of solving TSP with sizes that can reach hundreds or even thousands of vertices, while the maximum number of vertices contained in a VRP that can be solved is near a hundred vertices. Because the number of vertices in practices more often than never exceeds the specified constraint heuristics and metahueristics are more commonly used to solve VRP Laporte, 2009.

As it has been mentioned earlier in chapter 2.2 VRP solving algorithms are divided into three main categories each differs by the way it tackles the problem, and by the approach the algorithm takes to find the best solution (when possible).

2.2.1 Exact algorithms

Exact algorithms were first introduced for VRP almost 50 years ago with the basic branch-and-bound scheme that later evolved to the complex mathematical programming algorithms. Following is a general trace of their development.

2.2.1.1 Branch-and-bound algorithms

The original Branch-and-Bound algorithm was introduced in the year 1959 by Dantzig and Ramser, then it was slightly improved by Christofides and Eilon in 1969 (Laporte, 2009). In the latter algorithm, m is an input parameter, the given graph is then extended by $m-1$ artificial depots and setting inter-depot arc costs equal to infinity. Then an m-TSP is solved on this graph by branching on arcs, as by (Little et al. (1963)) TSP algorithm, except that the shortest tree bound is calculated at each note, instead of relaxed assignment problem. The side constraints of the VRP are being handled by simple fathoming rules. A new algorithm was proposed by Laporte, Mercure, and Nobert in 1986 in which an improvement on the branch-and-bound algorithm was made based on the modification of the Carpaneto and Toth in 1980 TSP algorithm (Laporte, 2009).
Probably the first paper ever to introduce the name "vehicle routing problem" was the one proposed Christofides in the year 1976. In the paper the branch-and-bound algorithm it describes branches rather than arcs, which resulted in a a search tree that has a limited depth of m but was rather wide. Nevertheless neither of the fore mentioned algorithms were of any success except for when used on small or easy instances (Laporte, 2009).
Various improvements were later introduced to the field, starting from the second lower bound which was based on the concept of q-route that's been put forward by Houck et al. (1980). Christofides, Mingozzi, and Toth (1981) could successfully solve instances that were $10 \leq n \leq 25$ with these lower bounds.
Later on Hadjiconstantinou, Christofides, and Mingozzi (1995) developed an improved branch-and-bound algorithm that was capable of solving $n \leq 50$. Fishes (1994) later was able to incorporate the k-DCT lower bound within a branch-and-cut algorithm for VRP with a restriction such as return trips between customers and depot were not allowed, the result was an algorithm capable of finding a route for up to 134 vertices (Laporte, 2009).

2.2.1.2 Dynamic programming

Unlike branch-and-bound, dynamic programming approach for solving VRP does not seem to have attracted much of attention for its development started and

ended within 10 years only.

Eilon, Watson-Gandy, and Christofides (1971) formulated the VRP as dynamic programming as follows. Let $c(S)$ be the optimal cost of a single vehicle route through the vertices of $S \subseteq V\backslash\{U\}$. The objective is to minimize $\sum_{r=1}^{m} > c(S_r)$ over all feasible partitions $\{S_1, ..., S_m\}$ of $V\backslash\{0\}$. Let $f_k(U)$ be the least cost achievable using k vehciles and delivering to a subset U of $V\backslash\{0\}$. Then

$$f_k(U) = \begin{cases} \{c(U) & (k = 1), \\ \min_{U^* \subseteq U \subseteq V\backslash\{0\}} & \{f_{k-1}(U \subseteq u^*) + c(U^*)\}(K > 1). \end{cases} \tag{2.1}$$

The solution cost is $f_m(V\backslash\{0\})$ and the optimal partition corresponds to the optimizing subsets in (2.1). The state space can be reduced by using the feasibility or dominance criteria. A state-space relaxation algorithm was applied by Christofides, Mingozzi, and Toth (1981). They reported their ability to solve instances with $10 \leq n \leq 25$.

2.2.1.3 Vehicle flow formulations and algorithms

Extending the classical TSP formulation of Dantzig, Fulkerson, and Johnson (1954), each of Laporte and Nobert (1983), and Laporte, Nobert, and Desrochers (1985) proposed a two-index vehicle flow formulations for the VRP (Laporte, 2009).

According to Neddef and Rinaldi (2002), VF model can be reinforced by including a set of inequalities. such as generalized capacity constraints, frame capacity constraints, VRP comb inequalities, and some other inequalities that are based on the stable set problem. The authors mentioned formally developed a branch-and-cut algorithm through which they were able to solve six instance ($22 \leq n \leq 45$) without branching, and nine others ($51 \leq n \leq 135$) with some branching. more recent branch-and-cut algorithms based on the VF can be found thought the work of Lysgaard, Letchford, and Eglese (2004) (Laporte, 2009).

A number of not-so-successful three-index vehicle flow formulations were also proposed. As proposed by the name they use three-indexed variables such as x_{ijk} which would be equal to 1 if and only if the arc (i, j) is traversed by vehicle k.

However, as mentioned earlier on, because of two-index vehicle flow formulations being more successful, three-index VF formulations did not catch as much interest. Examples can be found in the works of Golden, Magnanti, and Nguyen (1977), as well as Fisher and Jaikumar (1982) (Laporte, 2009).

2.2.1.4 Commodity flow formulations and algorithms

In commodity flow formulations, a set of variables y_{ij} (or y_{ijk} in the case of multi-vehicle solutions) define the load of a vehicle on a the arc (i, j). Gavish and Graves (1979) set an early example of it but with no computational results. Based on the TSP model of Finke, Claus, and Gunn (1984), a formulation was proposed by Baldacci, Hadjiconstantinou, and Mingozzi (2004). The formulation works on an extended graph $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V} = V \cup n+1$, $n+1$ is a copy of the depot, and $\bar{E} = E \cup (i, n+1) : i \in V$. A vehicle route is defined as a direced path from 0 to $n+1$. Binary variables x_{ij} will be equal to 1 only if edge (i, j) is used in the solution, variables y_{ij} represent the vehicle load on (i, j), and variables $y_{ji} = Q - y_{ij}$ represent the empty vehicle space on (i, j) whenever $x_{ij} = 1$.
The formulation is:

$$\textbf{(CF) minimize} \sum_{(i,j)\in\bar{E}} c_{ij}x_{ij} \tag{2.2a}$$

$$\textbf{subject to} \sum_{j\in\bar{V}} (y_{ji} - y_{ij}) = 2q_i \qquad (i \in V\backslash 0), \tag{2.2b}$$

$$\sum_{j\in V\backslash 0} y_{0j} = \sum_{i\in V\backslash 0} q_i, \tag{2.2c}$$

$$\sum_{j\in v\backslash 0} yj0 = mQ - \sum_{i\in V\backslash 0} q_i, \tag{2.2d}$$

$$\sum_{j\in V\backslash 0} y_{n+1,j} = mQ, \tag{2.2e}$$

$$y_{ij} + y_{ji} = Qx_{ij} \qquad ((i, j) \in \bar{E}), \tag{2.2f}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad (k \in V\backslash 0), \tag{2.2g}$$

$$y_{ij} \geq 0, y_{ji} \geq 0 \qquad ((i, j) \in \bar{E}), \tag{2.2h}$$

$$x_{ij} = 0, 1 \qquad ((i, j) \in \bar{E}). \tag{2.2i}$$

Constraints (2.2b)-(2.2e) and (2.2h) define consistent flows from 1 to $n+1$, constraints (2.2f) ensures that the y_{ij} and y_{ji} variables are feasible, and constraints (2.2g) are degree constraints.Using inequalities expressed in terms of x_{ij} variables, this problem was solved by branch-and-cut. Optimally, several instances were solved with $16 \leq n \leq 135$. However, the success rate was very high for cases where $30 \leq n \leq 60$ and $3 \leq m \leq 5$ (Laporte, 2009).

2.2.1.5 Set partitioning formulations and algorithms

Let r denote a route, let a_{ir} be a binary coefficient which value is equal to 1 only in the case of vertex $i \in V\backslash 0$ belongs to route r, let c^* be the optimal cost of route r, and let y_k be a binary variable which value would be 1 in the case of route r being in used in the optimal solution. The problem then can be defined as follows:

$$\textbf{(SP) minimize} \sum_r c_r^* y_r \tag{2.3a}$$

$$\textbf{subject to} \sum_r a_{ir} = 1 \ (i \in V\backslash 0), \tag{2.3b}$$

$$\sum_r y_r = m, \tag{2.3c}$$

$$y_r = 0, 1 \qquad (\text{all } r). \tag{2.3d}$$

By definition constraint (2.3c) is not part of the standard SP (Set Partitioning) formulation, but most authors use it for VRP.

The formulation mentioned above is considered a straightforward SP formulation for VRP and was proposed by Balinski and Quandt (1964).

Because of the large number of potential routes encountered in most nontrivial instances and the computing difficulty being c_r^* coefficients which in turn requires solving an exponential number of instances of an NP-hard problem, directly applying this formulation is not considered practical.

Later on a number of column generation algorithms were proposed to solve this problem, such as Rao and Zionts (1968) which does not seem to have been practically tested, it was succeeded by Foster and Ryan (1976) which generates routes by dynamic programming which again was not run to completion. Full column

generation algorithm was later on developed by Agarwal, Mathur, and Salkin (1989) which was able to solve instances with $15 \leq n \leq 25$ successfully at last.
Two of the most successful VRP algorithms that were developed in recent years make partial use of SP formulations. The first was proposed by Fukasawa et al. (2006), which combines SP and branch-and-cut-and-price algorithm, and it successfully solved instances containing u to 135 vertices.
The second algorithm was proposed by Baldacci, Christofides, and Mongozzi (2008), which again combines SP formulations with some inequalities valid for VF. In short, this algorithm uses SP some constraints, then compute lower bounds on the dual of the linear relaxation of this problem, obtained by applying three ascent heuristics. The final dual solution is used to generate a reduced problem containing columns of reduced cost between the upper bound and the lower bound achieved. The problem is then solved by an integer linear programming solver. The algorithm was capable of instances with $37 \leq n \leq 121$ and is slightly better than that of Fukasawa et al. (2006) (Laporte, 2009).

2.2.2 Classical heuristics

As it has been constructed earlier on, the most basic difference between exact algorithms and heuristics/metaheuristics. Although, exact algorithms have the ability to find optimal solutions, they only work on relatively small sets of data (134 instances at most), and the computational power needed for them is so high, therefor they need more time than the other two approaches.
Heuristics (and metaheuristics) sacrifice the optimal solution for a nearly optimal solution in exchange for better computational time, bigger sets of data and in general more convenience. The problem however is that the results although usually near optimal or are pretty good there is no way to prove that they are not bad (Pop et al., 2011).
Three classes of heuristics are going to be mentioned as follows:

- Constructive heuristics: Nearest Neighbor and a Clarke-Wright based heuristic.
- Improvement heuristics: String Cross (SC), String Exchange (SE), String Relocation (SR), and String Mix (SM).

2.2.2.1 Constructive heuristics

A constructive heuristic is a type of heuristics which starts with an empty solution and builds it from scratch step by step until a full solution is obtained. Koulamas, 1998

2.2.2.1.1 Nearest neighbor The algorithm runs as follows: First the algorithm will find the closest neighbor to the depot and visit it, from there it will visit the nearest unvisited neighbor as long as the route capacity has not yet exceeded that of the vehicle Q. When route capacity reaches the limit specified earlier Q, the algorithm will start again from the depot and create another route where again it will not visit vertices that has been visited by earlier routes. When all the vertices has been visited the algorithm will terminate.
The cons of this algorithm include easy implementation and relatively short execution time. It should be noted that due to the greedy nature of the algorithm, it tends to miss shorter routes some times. The complicity of the algorithm is $O(n^2)$. Pop et al., 2011.

2.2.2.1.2 A Clarke-Wright based heuristic algorithm Where the number of vehicles is a decision-variable usually the Clarke and Write algorithm is used and considered one of the more famous algorithms.
The algorithm consists of two steps as follows:
Step 1 (Savings computation): In this steps (as the name suggests) the algorithm simply calculates the possible savings of the routes, through the following way: For each $i \in V_l$ and $j \in V_p$, where $l \neq p$ and $l, p \in 1, ..., k$ compute:

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}. \tag{2.4}$$

As it can be seen from the equation (2.4) $s_{ij} \geq 0$ and $s_{ij} = s_{ji}$. The savings are then ordered in a decreasing order.
As a beginning the algorithm creates k routes $(0, i_l, 0), l \in 1, ..., k$ as folows for each cluster v_l the algorithm defines $c_{0i_l} = min\{c_{0j} | j \in V_l\}$.
By doing so there will be as many routes as there are clusters and the total distance would be:

$$d = c_{0i_1} + c_{0i_2} + ... + c_{0i_k}. \tag{2.5}$$

Step 2 (Route extension): Determine the savings s_{ui} or s_{jv} for each route $(0, i, ..., j, 0)$ which (savings) can be used to merge the current route with another one that ends with $(u, 0)$ or starts with $(0, v)$ for any $u \in V_l$ and $v \in V_p$ where $l \neq p$ and $l, p \in \{1, ..., k\}$ and V_l and V_p are clusters not visited by the route $(0, i, ..., j, 0)$. It should be noted that whenever more than one saving is available the algorithm will choose the one with the biggest impact on the total cost (biggest general reduction in cost).
The algorithm will keep repeating the operation until no further merges are available.
This algorithm is considered easy to implement and has a running time of $O(n^2 log\ n)$. (Pop et al., 2011)

2.2.2.2 Improvement heuristics

Improvement heuristics for VRP basically work on producing the best result possible by first proposing any set of routes and then improving the routes as the name suggests. The improvement process can be approached in two different manners, either (1) sequentially meaning that each route will be improved as a single route alone, or (2) in parallel in which case the improvements will be introduced to the graph as a whole and improvements can effect more than one route at each step (Pop et al., 2011).
Due to the complexity of the improvement process heuristics are used to propose the best routes possible, four of the proposed heuristics are as follow:

2.2.2.2.1 String cross (SC) In this method a string or chain is exchanged in sides between two routes as illustrated in figure (2.1).

2.2.2.2.2 String exchange (SE) Two strings one from each route with a maximum of k vertices in each string are exchanged between the two routes as shown in figure (2.2)

2.2.2.2.3 String relocation (SR) In this method a number of vertices k is moved to another route. Usually $k = 1$ or 2. An example is shown in figure (2.3)

2.2.2.2.4 String mix (SM) This method selects the methods that yields the best results out of SE and SR, to achieve that the algorithm takes two steps: **(1)** ***First Improvement (FI):*** applies the first move that can present an improvement to the objective. and **(2)** ***Best Improvement (BI):*** all possible moves are evaluated but only the one with the best improvement is applied (Pop et al., 2011).

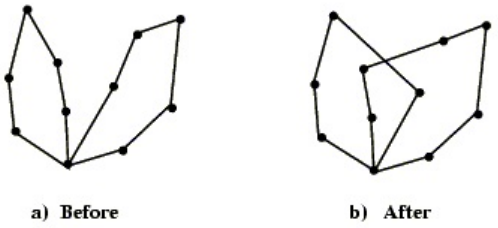

Figure 2.1: String Cross example

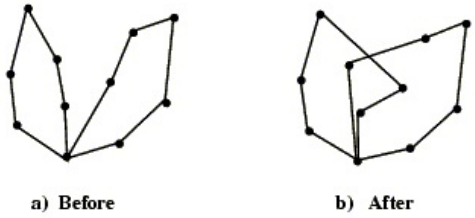

Figure 2.2: String Exchange example

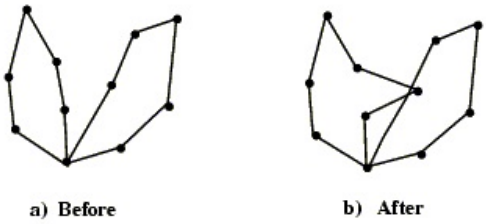

Figure 2.3: String Relocation example

2.2.3 Metaheuristics

Most metaheuristics can be regarded as improvement algorithms. While most of them perform well and are considered robust, they usually start from a low-quality solution first and then improve upon it (Laporte, 2009).

Following are some of the most popular metaheuristic algorithms when talking about VRP.

2.2.3.1 Ant colony optimization (ACO)

Inspired by ants in the real world, the ant colony optimization uses artificial ants to mimic the behavior of those in the real world.

Ants in nature leave traces of a chemical compound called the pheromone behind them on the ground, so when an ant is looking for a route it starts out by wandering randomly leaving behind it the trace of pheromone, each successor

ant will then either follow the pheromone (traces) it has found from previous ants or create its own new route on which it will leave new traces of pheromone with a degree of randomness that decreases the stronger the pheromone is. Since the shortest routes tend to accumulate more pheromone than others they would slowly become the favorite. Traces of pheromone decay with time, thus weakening less used routes and emitting them eventually, while consistent activities and usage of the favorite routes keep the pheromone fresh and at high rate on them.
As stated earlier the algorithm mimics nature in the way it works, where the artificial ants try to find the best solution starting randomly and trying to find better solutions from there. Each artificial ant starts from a point and with a specific rate of randomness either selects earlier discovered and visited solutions (next vertices in VRP) or starts a new route, the pheromone is represented by memory in the computer and each time a solution is visited again by a new ant its score is strengthened, at the same time on each cycle all of the solutions weakened at a similar rate to represent the effect of decaying pheromone in real life.
ACO is widely implemented in TSP (Traveling Salesman Problem) due to the similarity of the problem and the way ants react when fetching food. However ACO has been adapted to the VRP in recent years (Gendreau et al., 2008).

2.2.3.2 Genetic algorithms

Genetic algorithms are usually used to mimic the process of evolution of creatures in nature as described by the Darwinian principle of natural selection.
Simply put the algorithm starts from a population of solutions (often encoded as strings of bits which represent chromosomes) which improve by producing new generations over and over again, each generation improves upon the previous one using by either selection of the fittest, genetic crossover, or mutation. In each cycle only the fittest solutions are selected to have offspring solution thus making sure improving the probability of getting a better solution in the next generation. Moreover, other generation generating methods are used as well, such as mutation of a solution randomly or the selection of the fittest as well. The process is repeated a number of times and then the best solution is found and returned back as a result of the algorithm.

When GA is used with VRP, it is usually specifically encoded to accommodate the VRP (by either ignoring the solution encoding into chromosomes or applying the various operators directly on the solution) (Gendreau et al., 2008).

2.2.3.3 Greedy randomized adaptive search procedure

Basically what GRASP (Greedy Randomized Adaptive Search Procedure) does is using a randomized greedy heuristic to construct a variety of solution. At each step of the construction heuristic run, the solutions that has not yet been added to the partial solution group are evaluated with a heuristic function, the best of the evaluated solutions are added to a restricted solutions group called candidate list. From the candidate list one randomly chosen solution is added to the partial solution group.

The construction process is then completed and the solution is improved using a local search. After getting a final solution the algorithm is restarted a number of times to get a group of solutions just like mentioned before and then the best solution from the finalists (final solution of each restart) is selected and returned at the end (Gendreau et al., 2008).

2.2.3.4 Simulated annealing

Simulated Annealing is inspired by actions that are performed in the real physical world just like ACO (Ant Colony Optimization) as described in section (2.2.3.1). In physics Annealing can be defined as the process which aims at generating solids with low-energy states. This is achieved by first melting down the solid to liquid by increasing the temperature, and then gradually and carefully decreasing the temperature in a way that assures that the matter achieves equilibrium by holding the temperature at a specific level for a specific time carefully. The result of this process is a more regular structures associated with solids with low-energy states.

The algorithm on the other hand is basically a modification on the local search algorithm where the modification over the current search can be accepted with a degree of randomness in case of increasing the cost. The randomness is determined by a criterion called the temperature criterion (like physics annealing process).

The algorithm randomly modifies the current solution, in the case of the new solution having a lower cost, it is accepted as a current solution, otherwise, the probability of accepting it depends on the temperature criterion which has a value that decreases over iterations until it gets to the point where it no longer allows for cost increases to be accepted any more.
By implementing the principals mentioned above, SM avoids bad local optima, and with time it reaches the global optima as the temperature criterion's value becomes lower. Tests has shown that SA successfully traverses to global optima just like it is the case with other local search based algorithms.
The success of SA triggered development of similar algorithms like: Threshold Accepting, Record-to-Record Travel, and THe Great Deluge Algorithm which all had similar successful results (Gendreau et al., 2008).

2.2.3.5 Tabu search

TS (Tabu Search) is another algorithm based on local search like SA. It basically searches for the best solution in the current neighborhood, and moves it even if the cost increases. This allows the algorithm to avoid bad local optima cases as well. The algorithm holds a short-term memory where it holds a list of the solutions last visited and thus avoids visiting them again, by doing so the algorithm avoids short-term cycling.
The algorithm stops when reaches a specific number of iterations or when a specific number of searches fail to yield a solution with a better cost compared to the best known solution.

2.2.3.6 Variable neighborhood search

Just like Tabu Search, Simulated Annealing, Variable Neighborhood Search (VNS) is another metaheuristic algorithm based on local search.
This algorithm deploys a new approach to escape the bad local search optima by working on many multiple (often nested) neighborhoods at once. When a local optima is reached in a specific neighborhood another one is selected and used for the next iteration.
The algorithm then restarts from the first neighborhood when either all neigh-

borhoods had been tried or when no new better solution has been found.
Another known variation of VNB is Variable Neighborhood Descent (VND) (Gendreau et al., 2008).

3. SOLUTION

This chapter describes the algorithm we developed by details and points out the strengths and weaknesses of the algorithm.

3.1 Overview

The solution being discussed in this paper can be best described through its flowchart (figure: 3.1).
The flowchart describes how the algorithm works in a broad way, The algorithm starts with the sectorization which basically breaks down the graph into a number of smaller sectors which makes it possible to use exact methods to solve the routing problem in each sector. After the sectorization is done, routing algorithms are applied to find the best route in each of the sectors we obtained already in the first step.
The algorithm then evaluates the solution obtained and stores the evaluation result in the memory for later referencing. Afterwards, it checks the time window the program was given for execution, in the case of running out of time, the algorithm will search the memory looking for the best solution found (according to the evaluation results) and output it to the user, or else a resectorization process will be called to work on the sectors and change their combination in order to find a new combination that might have a better potential in being an optimum (or near optimum) solution.

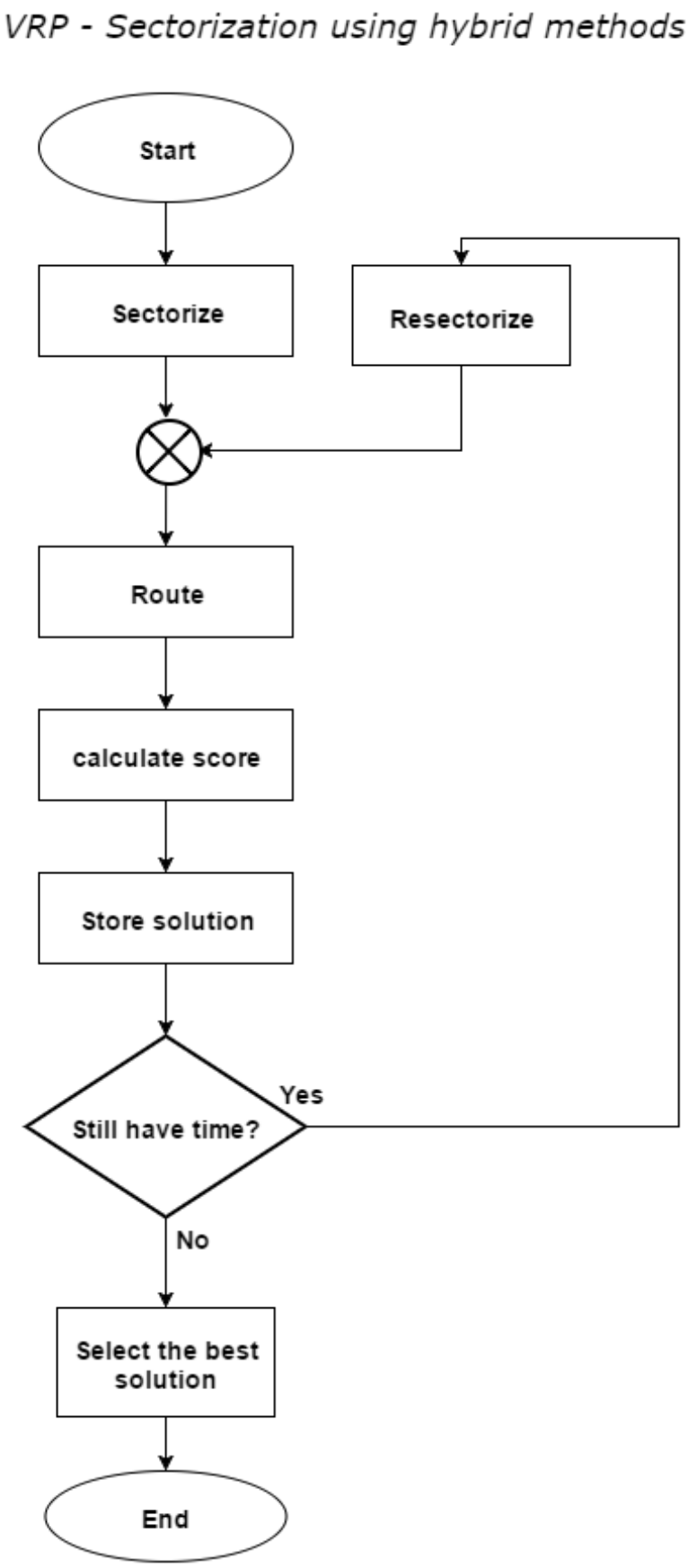

Figure 3.1: General flowchart for the whole solution

3.2 Sectorization

The whole solution is based on the first phase which is the sectorization, although it altered throughout the solution findings, but it is the milestone upon which everything is based and the better the initial sectors are, the faster it would be to get the best results.

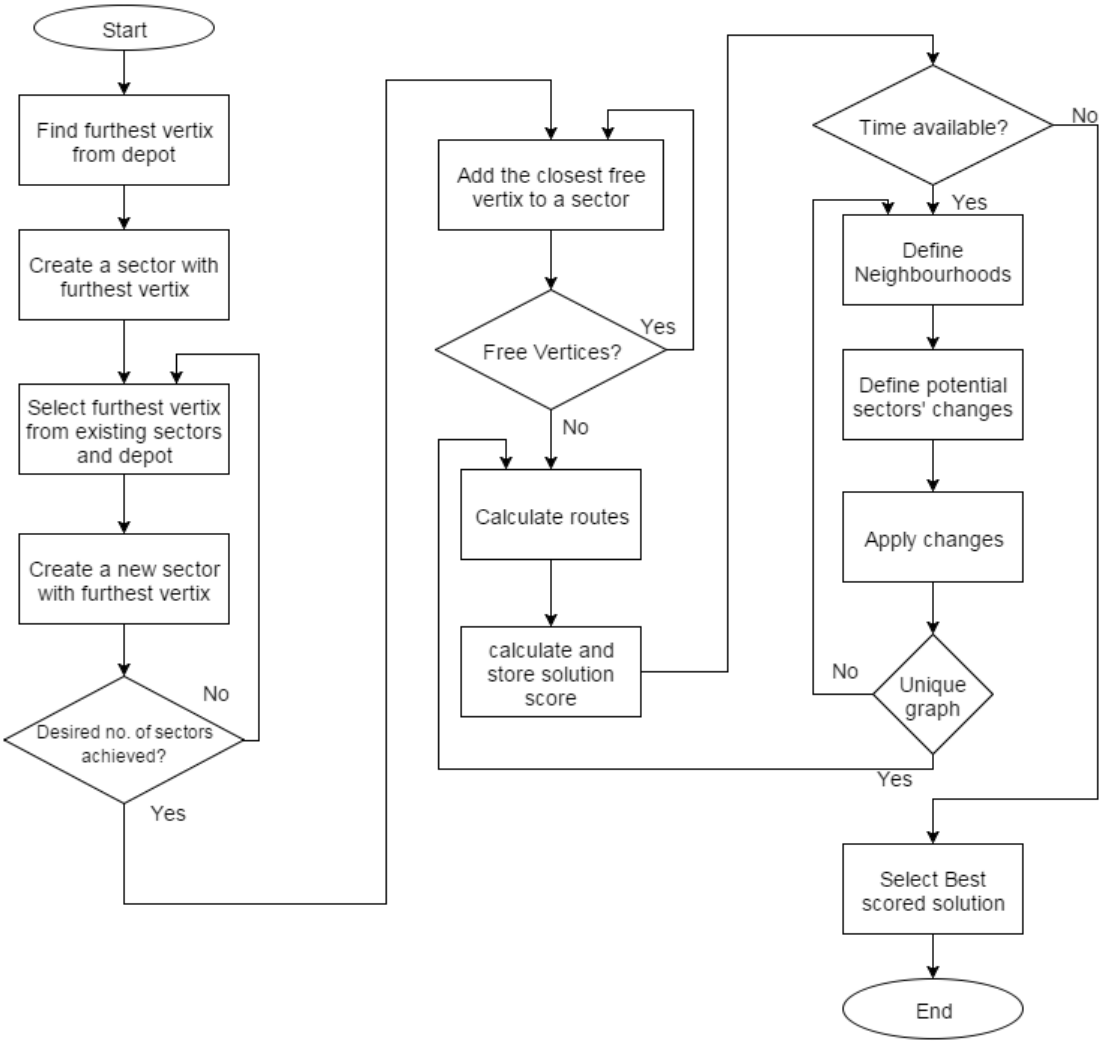

Figure 3.2: In-depth flowchart

3.2.1 Sector seeds selection

In order to find the best combination of sectors for each particular graph the program works on, it is vital to find a good starting point (seed) for each sector which is done by calling **MaxDist** method (Mourão et al., 2009).

MaxDist is an algorithm which tries to distribute the sectors along the graph in

a way which makes their seeds as far as possible from each other. The first sector seed is the vertex with the maximum *euclidean-distance* from the depot vertex.

$$\max_{1 \leq i \leq n} \sqrt{(V_{i_x} - V_{depot_x})^2 + (V_{i_y} - V_{depot_y})^2}$$

Next sector seed S_i will be the one with the biggest sum of *euclidean-distances* from the existing sector seeds and the depot

$$\max_{1 \leq i \leq n} \sum_{j=0}^{i-1} \sqrt{(V_{i_x} - V_{j_x})^2 + (V_{i_y} - V_{j_y})^2} + \sqrt{(V_{i_x} - V_{depot_x})^2 + (V_{i_y} - V_{depot_y})^2}$$

where s : number of initialized sectors

By the end of **sectorization initiation** process, the graph will have M sector seeds that are the furthest vertices from each other and the **depot** at the same time. Other than that the graph will have $N - M - 1$ free vertices (1 stands for the depot vertex) waiting to be obtained by the already-established sectors in the graph.

3.2.2 Obtaining free vertices

This process has the goal of leaving no free vertices in the graph by adding each free vertex to the sector that suits it best. It is this process's responsibility to make sure that no sector exceeds its size limit as well.
This is done by finding the smallest *euclidian-distance* between pairs of **free** and **sectorized** vertices and adding it to the corresponding sector and then repeating the same process over and over again until no free vertices are left in the graph.

$$\begin{aligned} \min_{d} \quad & d = \sqrt{(V_{i_x} - V_{j_x})^2} + \sqrt{()V_{i_y} - V_{j_y})^2} \\ \text{s.t.} \quad & Sector_k \leq SectorMaxSize \\ \text{where} \quad & V_i \in \text{free vertices} \\ & V_j \in \text{vertices included in sectors} \end{aligned}$$

By the end of this process the graph will have all of it vertices included in the sectors that were initialized in the previous step while maintaining the sizes of the sectors within the limit specified beforehand.

3.3 Routing

Routing phase comes right after the sectorization phase, which means that by the beginning of the routing phase, the program will have a set of sectors with a known start and end vertex (the depot). In other words, the program will have a set of directed graphs, and the application's job is to find the best route inside each of the graphs. If the program succeeds in finding the best combination of sectors and the best routes inside each of the sectors, it will be guaranteed that the current solution is an optimum one.

Given that the graphs which represent the sectors are significantly smaller in size than the whole graph (a sector cannot contain more than 130 vertices), it is viable to use exact methods to find the best routes, that way the application will have the optimal set of routes for any set of sectors, so the better the sectors distribution is, the better the solution will be.

The routing itself for each sector is considered an asymmetric traveling salesman problem (ATSP) in which the goal is to find the least cost Hamiltonian circuit (a circuit that traverse all the vertices in the graph), that's satisfied by solving the following formulas:

$$\textbf{minimize} \sum_{(i,j)\in A} C_{ij}X_{ij}, \tag{3.1a}$$

$$\textbf{subject to} \sum_{i\in V:(i,j)\in A} X_{ij} = 1 \forall j \in V, \tag{3.1b}$$

$$\sum_{j\in V:(i,j)\in A} X_{ij} = 1 \forall i \in V, \tag{3.1c}$$

$$(i,j) \in A : x_{ij} = 1 \tag{3.1d}$$

$$x_{ij} \in 0,1 \quad \forall (i,j) \in A, \tag{3.1e}$$

It must be noted that the constraint 3.1d has been added to make sure the tour does not contain subtours. Where x_{ij} is a binary variable that is equal to 1 if arc (i,j) is in the tour, or is equal to 0 otherwise. (Bekta?? and Gouveia, 2014).

A known extend of ATSP is the TMZ (Miller, Tucker, and Zemlin) alternation which was proposed in 1960 adding the following sub-tour elimination constraints:

$$U_i - I_j + (n-1)x_{ij} \leq n-2 \quad \forall i \neq j = 2, ..., n, \tag{3.2a}$$

$$U_i \in R \quad \forall i = 2, ..., n. \tag{3.2b}$$

Constraint 3.1d is replaced by 3.2a and 3.2b hence the new formula is referred to as *F(MTZ)*. The addition of the variable U_i in these constraints is used for ordering the vertices in the tour except for the depot to prevent the formation of illegal subtours. That's ensured by having $U_j \geq U_i + 1$ when $x_{ij} = 1$. Constraint 3.2b can be replaced with:

$$0 \leq u_i \leq n-2 \quad \forall i = 2, ..., n, \tag{3.3a}$$

The constraints 3.3a help find a precise meaning to each variable u_i, in other words, the value of u_i indicates the position of the vertex i in the tour, or to be precise, the number of vertices between the vertex i and the start vertex in the optimal tour.

An improvement over the MTZ constraints is the addition of the constraint 3.4a which was introduced by Desrochers and Laporte (1991) which replaces the 3.2a constraint

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 \forall i \neq j = 2, ..., n. \tag{3.4a}$$

The new constraint 3.4a serves two reasons: (i) it makes the relation between u_i and x_{ij} variables stronger because when $x_{ij} = 1$, a combined use of constraints 3.4a for pairs (i, j) and (j, i) imply $u_j = u_i + 1$, and (ii) they imply the CLIQUE constraints for sets S with two vertices. This is easy to see by adding two DL inequalities, one for a given pair (i, j) and the other for the reversed pair (j, i), which results in the CLIQUE constraint for the set i, j. (Bekta?? and Gouveia, 2014)

3.4 Sectors Optimization

The Optimization process is inspired by the ACO (Ant colony Optimization). However unlike conventional optimization algorithms, the optimization here is done to the sectors and their alignment in the graph rather than optimizing the routes themselves.

The idea is to give the program some freedom to randomly exchange vertices between sectors while keeping some influence over that freedom to shape it into a kind of experience that the program will gain with each execution iteration.

3.4.1 Optimization initiation

At the beginning of this phase the program will already have a viable solution which meets the criteria specified, but the solution might not be the best solution available. It should be noted that the optimality of the solution falls as the graph size and criteria associated with it rise in complexity. For this reason it is vital to try to optimize the solution that has been achieved in quest to find the optimal one.

This is done by specifying a connection strength between each vertex and sector in the graph (from now on it will be referred to as pheromone), the sum of all the pheromones associated with each vertex will be 1 at the initiation of the optimization given that the closer a vertex is to a sector's centroid the stronger the connection is hence the higher the pheromone value is.

By the end of this process the program will have a $2D$ pheromone matrix P that represents the pheromones needed for the optimization process to go on.

It should be noted that the **optimization initiation** process is done only once in the execution cycle of the program.

3.4.2 Optimization iterations

With each iteration the program will first go through all the sectors in the graph, and whenever there would be a sector with a size less than the size limit specified in the program it will try to acquire new vertices to that specific sector. For that to be possible two steps have to be processed: **(1) local**, and **(2) global**

3.4.2.1 Local

Local processes are done for each sector individually and they are done in the following order:

3.4.2.1.1 Neighborhood definition For a sector $s \in S$ to obtain a vertex $v \in V$, it must be an attractive move and that translates into the vertex v being close to the borders of sector s or included inside of them. That's where the idea of neighborhood definition originated from, the algorithm should be able to decide which vertices $v \in V$ are attractive and sound like a good move and which ones are not.

First the algorithm calculates the diameter (ϕ) of the sector (s) which is the maximum *euclidean-distance* between a vertex in the sector and the centroid of it.

$$\phi = \max \sqrt{(V_{i_x} - centroid_x)^2 + (V_{i_y} - centroid_y)^2}$$

After finding the diameter ϕ of the sector s, it is multiplied by neighborhood area percentage factor p which specifies whether the algorithm will try to minimize the area of sectors or let them grow in size to look for new possibilities outside of the current borders.

At that point the algorithm will try to look for vertices that lie within the neighborhood limits but are not already obtained by the sector s. However, vertices that lie within sectors with small sizes (smaller than a specified limit) will be excluded from the neighborhood.

Finding Z the set of vertices that represent attractive moves for the algorithm is considered the last step of **neighborhood definition**.

3.4.2.1.2 Experience and randomness In the case of the algorithm finding one or more attractive moves for the current sector s, it moves to the second step which is, combining the experience and knowledge we already have for this sector which is stored in the pheromone matrix P with randomness generated by the algorithm.

For each vertex $v \in Z$ in the neighborhood list the algorithm multiplies the pheromone from the pheromone matrix P that corresponds to the sector s and the

vertex v by a random number between 0 and 1. It then finds the biggest number among the results of the multiplications and selects it to obtain its vertex v to the sector s.
The last step (obtaining vertices) is repeated until the sector s obtains the number of vertices specified.
After the algorithm is done with obtaining vertices, the pheromone p corresponding for each vertex v and its respective sector is incremented by a fixed amount that is specified by the user variables.
At the end of each the process new vertices that have acquired are locked so they won't be analyzed for obtainment by other sectors at the same iteration.

3.4.2.2 Global

By the end of the previous steps the optimization iteration would come to an end and at that point the experience of the algorithm should be updated. This is done by decrementing each pheromones $p \in P$ by a fixed specific amount that is specified by the user variables beforehand.

3.5 Solution Inspection

The algorithm is a combination of EM (Exact Methods) and heuristics. One of EM's famous shortcomings is process time, which means that the algorithm process the EM parts only when needed.
One way to ensure that this is how the algorithm works is by inspecting each solution after each **optimization iteration** to find out whether it is a solution that needs the EM part to be processed or not.
The inspection has two steps: (1) finding whether this is a totally new and unique solution and (2) guessing by studying its properties that it can make up for a good solution.

3.5.1 Uniqueness inspection

This process finds out whether the solution it is working on is a totally new and unique solution that has never been studied before. This is achieved by first comparing the sizes of the sectors of the new solution by the sizes of sectors in

each of the old solutions. In case there are no equalities found, that would make for a new unique solution, otherwise the new solution move to next step.
Next step is deep comparison between each set of sectors in the new solution and each set of sectors in each of the previous solutions. Two sectors are identical if they both have the same set of vertices included in them, otherwise they are different.
For a solution T_i to be considered the same as an old solution T_j, all of the sectors of the old solution T_j must be identical to those of the new solution T_i.
If a solution passes the uniqueness inspection it will move to the next steps in the process, otherwise, it will be optimized over and over again until a unique solution is found.

3.5.2 Diameters inspection

After a solution passes the **uniqueness inspection** it moves to the **diameters inspection** which has the goal of guessing whether the new solution has the potential to be a good solution or not.
At first the new solution's sectors' diameters are calculated and then summed together and compared to the sum of diameters of the best solution available at until the moment, if the new solution has a combined size of diameters smaller or acceptably bigger (a percentage defined by the user variables) than the combined diameters of the best solution, it passes the diameter inspection and moves to the routing calculation step again. otherwise, it is considered a long shot and the time of finding the routes for it is saved in an attempt to find better solutions withing the given execution time-window.

3.6 Solutions Evaluation

The last step in the algorithm is finding the best solution out of the solutions that it has already found. This is done by calculating scores for each solution and then comparing them to one another.
It should be noted that the scores are being calculated according to the different criteria specified for the solution and the importance of each criterion. The algorithm can as well find out which solution is the best if the user decides to take

only one criteria into consideration.

After the algorithm calculates the routes of each solution it sums the cost of traversing each route from the depot and making the full circle back to the depot for each type of costs. Afterwards it sums the costs of the same type for all routes. It then calculates a combined cost which is a defined as the weighted cost of the routes combined (given that the user defines the weights of costs).

The solution with the lowest combined cost is the global best solution that the algorithm has achieved. other best solutions will be available as well, which are the best solutions according to each of the costs specified for the algorithm beforehand.

It should be noted that due to the nature of the algorithm (demanding time), the algorithm keeps updating its best solution array to be able to output the data at anytime in case of emergencies.

4. TESTS AND BENCHMARKS

This chapter we will focus on the results of running the algorithm in real life and how it performs when benchmarked against other established and tested algorithms in similar environments.

4.1 Test Data

For reasons related to availability of data and test results, Turkish cities were selected for testing and benchmarking the algorithm.

4.1.1 Assumptions

To make sure the data fits into the use cases of the algorithm the following assumptions were made:

- Each city center represents one client that must be visited
- Each client (city center in this case) must be visited once and only once
- Ankara city center is considered the depot from which all the trips should originate from and end at.
- The trips are going to made by 8 vehicles
- Each client (city in this case) is going to be visited by one vehicle only.
- To represent a payload limit, each vehicle is constrained to visiting 12 clients at most.
- The only cost associated with trips between the depot and clients or between clients is the distance.
- Distance importance (weight) is 100 percent.

4.1.2 Technical specifications

For testing, a laptop with the following specifications is being used:

- CPU: Intel Core i5 2350U, 2.3GHz, 3MB cache
- RAM: 8GB DDR3
- Storage: 250GB SSD
- Graphics: AMD Radeon Graphics 3660 w/1GB of DDR3 VRAM
- OS: Microsoft Windows 7 64-bit
- Softwares: Python 2.3, IBM CPLEX Studio 12.6, Microsoft Office 2016

4.1.3 Used data

For the algorithm to work as designed, different types of data should be supplied, including the coordinates of clients, drive distances between them, and other cost related parameters.
Data fed to the algorithm is split into different essential parts, starting from Turkish cities coordinates, which are specified in (table A.1).
In order to yield realistic results, the algorithm is fed distances between each pair of the cities specified in the table above as well.
The algorithm can be fed with data that describes different types of expenditures and costs as well as the importance (weight) of each one of them given that all of the weights added would sum up to 1. In this case the cost was simply the distances between cities and the weight was set to 1 which indicates that the only cost would be the distance and its importance is 100%.
The distances between all the cities in the Republic of Turkey are included in the attachments in an excel file called "Turkish_cities_distances.xlsx" (General Directorate of Highways in Turkey, 2016).

4.2 Results

The algorithm was tested using two different sets configurations; (1) short flexible configurations, and (2) long strict configurations.

4.2.1 Short flexible configurations

For this type of execution, configurations were set as follows:

- **Number of vehicles (sectors):** 8
- **Number of optimization iterations:** 280
- **Neighborhood min size during optimizations:** 60%
- **Neighborhood max size during optimizations:** 1000%
- **Local pheromone update value:** 0.05
- **Global pheromone update value:** -0.03
- **Minimum number of clients per vehicle:** 8
- **Maximum number of clients per vehicle:** 11

Completing **280** optimization iterations took the test environment described in section 4.1.2 about **128** minutes of execution, which translates into an average of **27.45** seconds per iteration and it yielded the best suggested solution at iteration **#61** with a solution total cost (total distance traversed by all vehicles) of **18,656KM**. The total cost is broken down into the following details per vehicle:

- **Vehicle #1:** 3428KM
- **Vehicle #2:** 3058KM
- **Vehicle #3:** 2274KM
- **Vehicle #4:** 1357KM
- **Vehicle #5:** 1918KM
- **Vehicle #6:** 1608KM
- **Vehicle #7:** 2624KM
- **Vehicle #8:** 2389KM

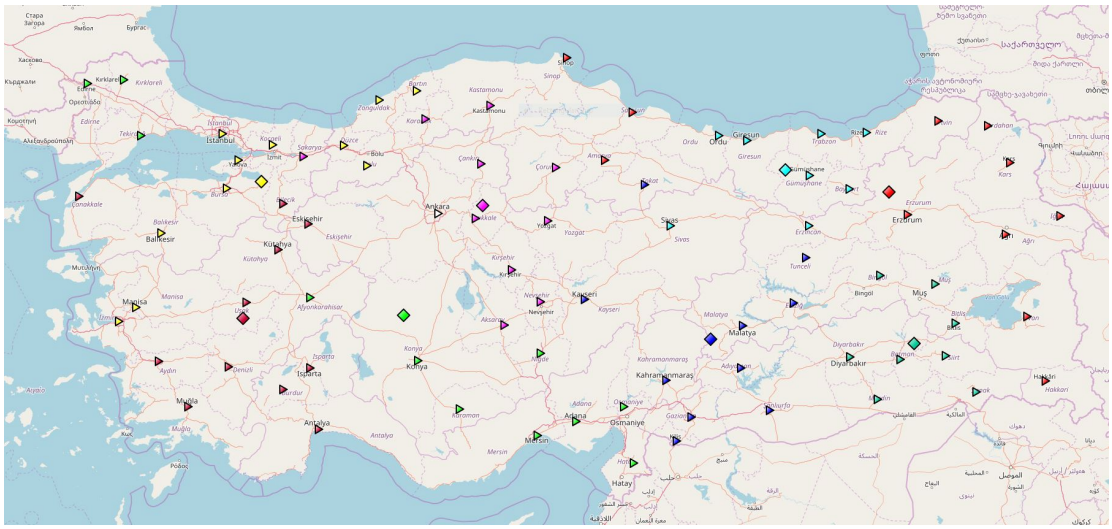

Figure 4.1: Short test's best solution's clients distribution per vehicle

Calculating the standard deviation (SD) of the results, it is concluded that $\sigma \simeq 658KM$. The reason of such a big deviation can be caused by multiple reasons, most importantly are: **(1)** the lack to iterations, and **(2)** flexibility of configurations (Neighborhood max size of **1000%** is too flexible), and **(3)** the lack of experience fed into the algorithm.

In order to avoid bad results, multiple measures can be taken such as increasing the number of optimization iterations, train the algorithm to increase its experience thus increasing the efficiency, and including different factors of the solution in the algorithm's input matrices.

Inspecting the map in figure 4.1 it can be seen that the algorithm failed to group the clients in the best possible way (due to the lack of time).

4.2.2 Long strict configurations

To try to overcome the limitations of the algorithm when tried with limited number of iterations, and other relaxed configurations, the algorithm was tested again on the same environment but with the following configurations:

- **Number of vehicles (sectors):** 8
- **Number of optimization iterations:** 500
- **Neighborhood min size during optimizations:** 60%
- **Neighborhood max size during optimizations:** 150%

- **Local pheromone update value:** 0.05
- **Global pheromone update value:** -0.03
- **Minimum number of clients per vehicle:** 8
- **Maximum number of clients per vehicle:** 11

Completing the **500** optimization iterations took the test machine **357** Minutes, resulting in a mean of **42** per iteration. best suggested solution was solution **#337** and had a total cost of **17,277KM** which translates into a **7.4%** decrease in costs when compared to the previous test (described in section 4.2.1), when compared to the increase in processing time, we conclude that the costs decreased **1.8% per processing hour**. Below are the details of the best suggested solution:

- **Vehicle #1:** 2802KM
- **Vehicle #2:** 1721KM
- **Vehicle #3:** 1803KM
- **Vehicle #4:** 1561KM
- **Vehicle #5:** 2961KM
- **Vehicle #6:** 1745KM
- **Vehicle #7:** 2898KM
- **Vehicle #8:** 1786KM

Calculating the standard deviation (SD) for the results above it is concluded that $\sigma \simeq 582KM$ which quiet lower than the previous trial (section 4.2.2) with an improvement of **11.6%**.
Studying the map in figure 4.2 shows that the algorithm successfully grouped the clients in tighter groups which resulted in the small deviation in distances traveled for different vehicles and lower solution-wide cost as well.

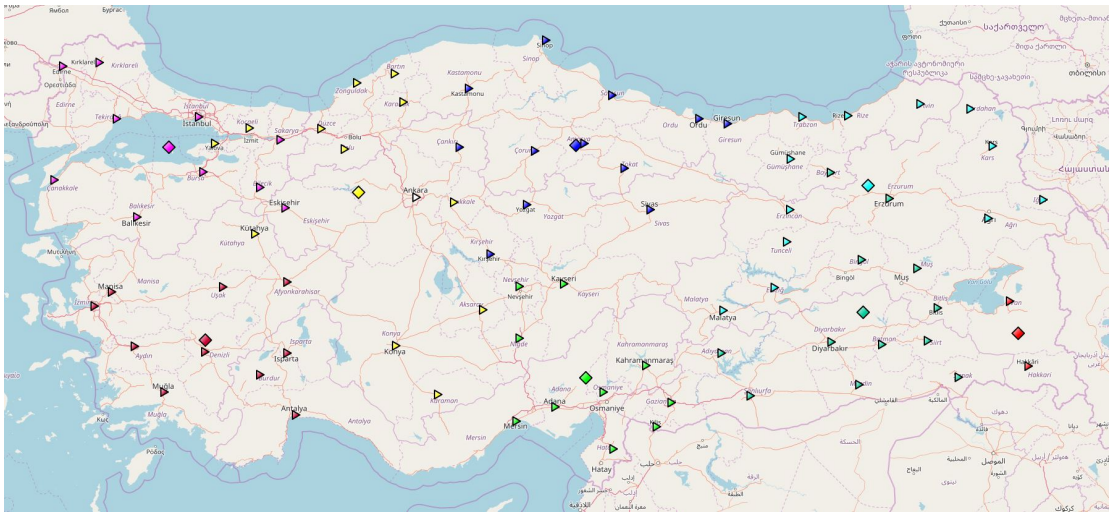

Figure 4.2: Long test's best solution's clients distribution per vehicle

4.3 Benchmarking

In a world where VRP is a problem such a high importance, it is important to benchmark new solutions against other established ones. Therefor, the algorithm is being benchmarked against "VRP Spreadsheet Solver" which was developed using MS-Excel by Dr. Güneş Erdoğan in 2013 (Erdoğan, 2013).
Both the tool (VRP Spreadsheet Solver), and our algorithm were fed the same data, set to run using the same configurations and were given exactly the same time (357 Minutes) to run on the same environment (same PC and same softwares), and the results were as follow:

- **Vehicle #1:** 1969KM
- **Vehicle #2:** 907KM
- **Vehicle #3:** 1017KM
- **Vehicle #4:** 1980KM
- **Vehicle #5:** 2709KM
- **Vehicle #6:** 1837KM
- **Vehicle #7:** 1857KM
- **Vehicle #8:** 2980KM

Summing up the previously provided individual costs will give a sum of **15256KM** of total cost for the solution provided by the tool, with a **671KM (14.9%)** standard deviation percentage. Comparing it to the total cost of the solution yielded by our algorithm, it is concluded that VRP Spreadsheet Solver yielded a solution **11.7%** cheaper when given the same configurations and the same time window for processing.
Benchmarking the algorithm against the tool (VRP Spreadsheet Solver) shows that the algorithm is not yet as accurate as other tools that are already in use.

4.3.1 Ways to improve and take advantage of the algorithm

It is suspected that the algorithm could have performed better using different set of configurations or in different scenarios as detailed in the following points:

- **Population size:** The set of clients that was provided for benchmarking the algorithm was too small (81 clients) to take advantage of the algorithm's usage of metaheuristics. As it has been stated earlier in chapter 2.2.1.1 algorithms based on exact methods are capable of solving problems that have a population of up to 134 instances.

- **Little expenses information:** The benchmarking did not take full advantage of the algorithm's ability of finding a solution that takes into account an infinite number of different expenses (in theory) due to the lack of real life data. The algorithm was only given 1 type of costs (distances between clients) and thus it lacked one of the advantages it possesses.

- **Runtime experience configuration:** The algorithm was designed to gain experience throughout its runtime, through the usage of ACO (Ant Colony Optimization) as explained in section 3.4.2.1.2, but to make further usage of experience, more testing is needed using real life data.

- **Algorithm lifetime experience:** As mentioned before in section 3.4.2.1.2, the algorithm makes use of experience within a runtime, but it can be improved to make of longer experience through the software lifetime by using the experience gained from one runtime in the future ones until the

algorithm has been trained enough to suggest solutions closer to the optimal solution in a shorter time window.

5. CONCLUSIONS AND RECOMMENDATIONS

This chapter discusses fields of application, conclusions, and recommendations for future improvements.

5.1 Practical Applications of the Algorithm

The algorithm can be used by almost all types of business around the world, such as logistic companies which are considered the main beneficiary of such studies, post services providers, and all other businesses that are looking for ways to distribute different amounts of goods to clients, and/or are looking for ways to more effectively manage their supply chain, it can also be used by to manage picking up clients or employees by HR (human resources) departments in companies or by hotels that offer pick services for their clients.

5.2 Conclusions

VRP (Vehicle Routing Problem) is a problem that has a huge impact over our lives as it has been stated in section 1.6, and the importance of this impact is shown in the number of studies that were done in the field. With such a big impact, a solution with the smallest amount of improvements would have an effect so big as the impact.

The importance that VRP gained through time made it the title of countless studies since its definition in the late 1950s. Many approaches were taken to solve this important issue since then including exact methods, classical heuristics and metaheuristics (as it was detailed in section 2.2) in pursuit of finding the optimal solution which in turn would help us be more efficient and thus reserve resources (human, natural, time,.. etc.).

In this thesis a new approach was introduced to solve VRP using a combination of exact methods and metaheuristics to harvest the benefits of both approaches while avoiding their shortcomings.

Metaheuristics were used to divide the population into sections in accordance with the number of the available vehicles and keep trying to find the best combination of sectors that would yield the best global solution (solution with the lowest cost) through experience gained while processing the algorithm.

Whereas exact methods were used to find the best route within each sector which insures that the best route within the section was suggested.

When benchmarked against VRP Spreadsheet Solver which is a tool that was developed by Dr. Güneş Erdoğan to solve the same problem, the algorithm performance was lower than that of VRP Spreadsheet Solver. The performance disadvantage is suspected to be caused by the small number of clients used in the benchmarking process and the lack of different expenses and criteria for costs for traversing between clients, which led to the algorithm not being able to take advantage of its strong points as it was stated in section 4.3. To take advantage of the algorithm more testing is needed using bigger number of clients and a more comprehensive set of expenses that would more clearly describe real life situations.

5.3 Future Improvements

Future improvements can be applied to the algorithm by adding constraints that would prohibit the usage of metaheuristics for problems with small number of clients, as well as adding the ability to change criteria in real time which would significantly improve the algorithm since it is an algorithm that takes lots of time to fully process and thus changes can be so costly sometimes.

Another field were the algorithm can be improved is by adding lifetime machine learning to the algorithm. This way previous solutions or trials would be used as an experience for the algorithm and thus yield in better solutions with shorter processing times in the future.

REFERENCES

Alba, R. D. (1973), `A graph-theoretic definition of a sociometric clique', *The Journal of Mathematical Sociology* 3(1), 113-126.

Amazon (2017), `Amazon.co.uk / About One-Day Delivery'. URL: https://www.amazon.co.uk/gp/help/customer/display.html?n

Bektaş, T. and Gouveia, L. (2014), `Requiem for the Miller-Tucker-Zemlin subtour elimination constraints?', *European Journal of Operational Research* 236(3), 820-832.

Dunn, J. (2017), The number of Amazon Prime members has reportedly doubled in the past two years, *Technical report*. URL: http://www.businessinsider.com/how-many-amazon-prime-subscribers-estimates-chart-2017-4

Erdoğan, G. (2013), `User's Manual for VRP Spreadsheet Solver', p. 18. URL: http://verolog.deis.unibo.it/vrp-spreadsheet-solver

Gendreau, M., Potvin, J. Y., Bräysy, O., Hasle, G. and Løkketangen, A. (2008), `Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography', *Operations Research/ Computer Science Interfaces Series* 43(August), 143-169.

General Directorate of Highways in Turkey (2016), `Distances Between Turkish Cities'. URL: http://www.kgm.gov.tr/sayfalar/kgm/sitetr/root/uzakliklar.aspx

Koh, K. M., Dong, F. M. and Tay, E. G. (2007), Introduction to Graph Theory: H3 Mathematics, *World Scientific*. URL: https://books.google.com/books?id=7 bQa4SJTQQC&pgis=1

Koulamas, C. (1998), `A new constructive heuristic for the flowshop scheduling problem', *European Journal of Operational Research* 105(1), 66-71.

Laporte, G. (2009), `Fifty Years of Vehicle Routing', *Transportation Science* 43(4), 408-416.

Mckinnon, A. (2007), `CO 2 Emissions from Freight Transport: An Analysis of UK Data', (iii).

Mouràò, M. C., Nunes, A. C. and Prins, C. (2009), `Heuristic methods for the sectoring arc routing problem', *European Journal of Operational Research* 196(3), 856-868. URL: http://dx.doi.org/10.1016/j.ejor.2008.04.025

Pop, P., Sitar, C., Zelina, I., Lupşe, V. and Chira, C. (2011), `Heuristic Algorithms for Solving the Generalized Vehicle Routing Problem', *Int. J. of Computers, Communications & Control* 6(1), 158-165.

PwC (2015), `Transportation and Logistics 2030-Volume 5: Winning the talent race', *Transportation & Logistics 2030* 5.

Reuter, J. (2010), `Transportation & Logistics 2030 - Transport infra-structure', Transportation 1, 1-64.

Ruske, K.-D., Kauschke, P., Basu, G., Reuter, J. and Montgomery, D. E. (2011), `Transportation & Logistics 2030 Volume 4: Securing the supply chain', *PricewaterhouseCoopers* 4, 50. URL: www.pwc.com/tl2030

United Nations (2015), `World Population Prospect: The 2015 Revision, World Population 2015 Wallchart.', *Department of Economic and Social Affairs, Population Division* p. 2. URL: https://esa.un.org/unpd/wpp/Publications/Files/World_Population_2015_Wallchart.pdf

World Trade Organization (2016), `World Trade Statistical Review', *World Trade Organization*.

Yeun, L. C., Ismail, W. a. N. R., Omar, K. and Zirour, M. (2008), `Vehicle Routing Problem: Models and Solutions', *Journal of Quality Measurement and Analysis* 4(1), 205-218.

APPENDICES

APPENDIX A.1: Turkish Cities' Coordinate

Table A.1: Turkish Cities' Coordinate

City No.	City Name	lat	long
1	ADANA	37.000000	35.321333
2	ADIYAMAN	37.764751	38.278561
3	AFYONKARAHİSAR	38.750714	30.556692
4	AĞRI	39.626922	43.021596
5	AKSARAY	38.368690	34.036980
6	AMASYA	40.649910	35.835320
7	ANKARA	39.920770	32.854110
8	ANTALYA	36.884140	30.705630
9	ARDAHAN	41.110481	42.702171
10	ARTVİN	41.182770	41.818292
11	AYDIN	37.856041	27.841631
12	BALIKESİR	39.648369	27.882610
13	BARTIN	41.581051	32.460979
14	BATMAN	37.881168	41.135090
15	BAYBURT	40.255169	40.224880
16	BİLECİK	40.056656	30.066524
17	BİNGÖL	39.062635	40.769610
18	BİTLİS	38.393799	42.123180
19	BOLU	40.575977	31.578809
20	BURDUR	37.461267	30.066524
21	BURSA	40.266864	29.063448
22	ÇANAKKALE	40.155312	26.414160
23	ÇANKIRI	40.601343	33.613421
24	ÇORUM	40.550556	34.955556
25	DENİZLİ	37.776520	29.086390
26	DİYARBAKIR	37.914410	40.230629
27	DÜZCE	40.843849	31.156540
28	EDİRNE	41.681808	26.562269
29	ELAZIĞ	38.680969	39.226398
30	ERZİNCAN	39.750000	39.500000
31	ERZURUM	39.900000	41.270000
32	ESKİŞEHİR	39.776667	30.520556
33	GAZİANTEP	37.066220	37.383320
34	GİRESUN	40.912811	38.389530
35	GÜMÜŞHANE	40.438588	39.508556
36	HAKKARİ	37.583333	43.733333
37	HATAY	36.401849	36.349810
38	IĞDIR	39.887984	44.004836
39	ISPARTA	37.764771	30.556561
40	İSTANBUL	41.005270	28.976960
41	İZMİR	38.418850	27.128720
42	KAHRAMANMARAŞ	37.585831	36.937149
43	KARABÜK	41.206100	32.620350
44	KARAMAN	37.175930	33.228748

City No.	City Name	lat	long
45	KARS	40.616667	43.100000
46	KASTAMONU	41.388710	33.782730
47	KAYSERİ	38.731220	35.478729
48	KIRIKKALE	39.846821	33.515251
49	KIRKLARELİ	41.733333	27.216667
50	KIRŞEHİR	39.142490	34.170910
51	KİLİS	36.718399	37.121220
52	KOCAELİ	40.853270	29.881520
53	KONYA	37.866667	32.483333
54	KÜTAHYA	39.416667	29.983333
55	MALATYA	38.355190	38.309460
56	MANİSA	38.619099	27.428921
57	MARDİN	37.321163	40.724477
58	MERSİN	36.800000	34.633333
59	MUĞLA	37.215278	28.363611
60	MUŞ	38.946189	41.753893
61	NEVŞEHİR	38.693940	34.685651
62	NİĞDE	37.966667	34.683333
63	ORDU	40.983879	37.876411
64	OSMANİYE	37.213026	36.176261
65	RİZE	41.020050	40.523449
66	SAKARYA	40.693997	30.435763
67	SAMSUN	41.292782	36.331280
68	SİİRT	37.933333	41.950000
69	SİNOP	42.023140	35.153069
70	SİVAS	39.747662	37.017879
71	ŞANLIURFA	37.159149	38.796909
72	ŞIRNAK	37.418748	42.491834
73	TEKİRDAĞ	40.983333	27.516667
74	TOKAT	40.316667	36.550000
75	TRABZON	41.001450	39.717800
76	TUNCELİ	39.307355	39.438778
77	UŞAK	38.682301	29.408190
78	VAN	38.489140	43.408890
79	YALOVA	40.650000	29.266667
80	YOZGAT	39.818081	34.814690
81	ZONGULDAK	41.456409	31.798731

RESUME

Mohammed Aref MANSOUR
Aleppo, Syria 1990
mhmdmnsr89@gmail.com

EDUCATION:

- **Bachelor :** 2014, Al-Shahbaa Private University, Faculty of Information Engineering, Department of Information Technology.

- **Master :** 2018, Istanbul Aydin University, Institute of Social Sciences, Department of Business, Business Administration Program.

PROFESSIONAL EXPERIENCE:

- **Project Manager and Technical Coordinator**
  Cemete Electronic Money and Payment Services INC., Istanbul Turkey
  Aug 2016 - Feb 2018

- **Foreign Customers Representative UniversalGYO**
  Istanbul, Turkey
  Feb 2015 - Apr 2015

- **Co-founder and Customer Service Manager**
  Inspiria INC., Aleppo, Syria
  May 2011 - Jan 2014

TRAINING:

- **ASSISTANT RESEARCHER (ERASMUS+)**
  INSESC TEC, Porto, Portugal
  Oct 2015 - Feb 2016

- **PHP TRAINING**
  Syrian-German Company for Internet Solutions, Aleppo, Syria
  Jul 2011

LANGUAGES:

- **Arabic:** Mother Tongue
- **English:** Proficient User
- **Turkish:** Proficient User