ROUTE OPTIMIZATION FOR SOLID WASTE TRANSPORTATION USING PARALLEL HYBRID GENETIC ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELİM ONUR UŞKAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF GEODETIC AND GEOGRAPHICAL INFORMATION TECHNOLOGIES

DECEMBER 2010

Approval of the thesis:

ROUTE OPTIMIZATION FOR SOLID WASTE TRANSPORTATION USING PARALLEL HYBRID GENETIC ALGORITHMS

Submitted by **SELİM ONUR UŞKAY** in partial fulfillment of the requirements for the degree of **Master of Science in Geodetic and Geographical Information Technologies Department, Middle East Technical University** by,

Date: 29.12.2010

ABSTRACT

ROUTE OPTIMIZATION FOR SOLID WASTE TRANSPORTATION USING PARALLEL HYBRID GENETIC ALGORITHMS

Uşkay, Selim Onur

M.Sc., Department of Geodetic and Geographical Information Technologies

Supervisor: Assoc. Prof. Dr. Ayşegül Aksoy

Co-Supervisor: Prof. Dr. Şebnem Düzgün

December 2010, 113 pages

The transportation phase of solid waste management is highly critical as it may constitute approximately 60 to 75 percent of the total cost. Therefore, even a small amount of improvement in the collection operation can result in a significant saving in the overall cost. Despite the fact that there exist a considerable amount of studies on Vehicle Routing Problem (VRP), a vast majority of the existing studies are not integrated with GIS and hence they do not consider the path constraints of real road networks for waste collection such as one-way roads and U-Turns. This study involves the development of computer software that optimizes the waste collection routes for solid waste transportation considering the path constraints and road gradients. In this study, two different routing models are proposed. The aim of the first model is to minimize the total distance travelled whereas that of the second model is to minimize the total fuel consumption that depends on the loading conditions of the truck and the road gradient. A comparison is made between these two approaches. It is expected that the two approaches generate routes having different characteristics. The obtained results are satisfactory. The distance optimization model generates routes that are shorter in length whereas the fuel consumption optimization model generates routes that are slightly higher in length but provides waste collection on steeply inclined roads with lower truck load. The resultant routes are demonstrated on a 3D terrain view.

Keywords: Solid Waste Transportation, Vehicle Routing Problem, Hybrid Genetic Algorithms.

ÖZ

PARALEL HİBRİT GENETİK ALGORİTMALARLA KATI ATIK TAŞINMASI İÇİN ROTA OPTİMİZASYONU

Uşkay, Selim Onur

Yüksek Lisans, Jeodetik ve Coğrafi Bilgi Teknolojileri

Tez Danışmanı: Doç. Dr. Ayşegül Aksoy

Ortak Tez Danışmanı: Prof. Dr. Şebnem Düzgün

Aralık 2010, 113 sayfa

Katı atık yönetimindeki taşıma safhası toplam maliyetin yaklaşık yüzde 60'ı ile 75'i arasındaki bölümü oluşturabileceğinden oldukça kritiktir. Bu nedenle, atık toplama aşamasındaki küçük çaplı bir ilerleme bile toplam maliyeti önemli miktarlarda düşürebilir. Her ne kadar Araç Rotalama Problemi (ARP) üzerinde oldukça fazla sayıda çalışma olsa da, mevcut çalışmaların çok büyük bir bölümü Coğrafi Bilgi Sistemi (CBS) ortamına entegre değildir ve dolayısıyla tek yönlü yollar ve U-dönüşleri gibi gerçek yol ağlarına ait kısıtlamaları dikkate almazlar. Bu çalışmada katı atık taşınması için atık toplama rotalarını optimize eden ve araç yükü ile yol eğiminin yakıt tüketimine etkisini de hesaba katan bir bilgisayar yazılımı geliştirilmiştir. İki farklı rotalama modeli önerilmiştir. İlk modelin amacı toplam katedilen mesafeyi, ikinci modelinki ise aracın yük durumu ve yol eğimine bağlı olan yakıt tüketimini minimize etmektir. İki yaklaşım arasında bir karşılaştırma yapılmıştır. İki modelin birbirinden farklı özelliklerde rotalar üretmesi beklenmektedir. Elde edilen sonuçlar tatmin edicidir. Mesafeyi optimize eden model daha kısa rotalar üretirken yakıt tüketimini optimize eden model biraz daha uzun fakat yüksek eğilimli yolları daha az yükle kateden rotalar üretmektedir. Elde edilen rotalar 3 boyutlu arazi modeli üzerinde gösterilmektedir.

Anahtar Kelimeler: Katı Atık Taşınması, Araç Rotalama Problemi, Hibrit Genetik Algoritmalar.

To My Family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Ayşegül AKSOY and my co-supervisor Prof. Dr. Şebnem DÜZGÜN for their valuable comments and guidance throughout the research. It was a great pleasure for me to carry out this thesis under their supervision.

I am deeply indebted to my family for their support and encouragement to accomplish this work.

At last, but definitely not least, I offer special thanks to my friends for their moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

GIS	Geographical Information Systems
VRP	Vehicle Routing Problem
DEM	Digital Elevation Model
GA	Genetic Algorithm
TSP	Traveling Salesman Problem
SWM	Solid Waste Management
PGA	Parallel Genetic Algorithm
ERX	Edge Recombination Crossover
MFC	Microsoft Foundation Classes
UTM	Universal Transverse Mercator

CHAPTER 1

INTRODUCTION

Solid Waste Management is a complex process involving many stages such as generation, on-site handling and storage, collection, transfer and transportation, processing and disposal of solid wastes (Nemerow et al., 2008). According to Nemerow et al. (2008), approximately 60% to 75% of the total solid waste management cost is spent for the collection phase, in which the waste is collected from the source and carried to the transfer station or landfill. Therefore, even a small improvement in the collection operation by selecting less-costly routes can result in a significant saving in overall cost. This issue becomes highly critical considering the high costs of fuel and labour.

Despite the fact that there exist a number of studies on Travelling Salesman Problem (TSP) and different variations of Vehicle Routing Problems (VRP) in the existing literature, the amount of studies on solid waste collection route optimization is limited. Moreover, a vast majority of the work done about vehicle routing is limited to hypothetical networks and are not integrated to Geographical Information Systems (GIS). Solid Waste Collection requires a new routing model with spatial constraints that represent the path conditions. It is important that the generated routes comply with the road directions and the U-Turns are avoided (Vesilind et al., 2001). In this point of view, GIS integration provides the means for the consideration of spatial constraints in the routing model.

Geographical Information Systems (GISs) are computer-assisted systems that are commonly utilized to capture, store, retrieve, analyze and display spatial data (Clark, 1986). By using a GIS software, it is possible to work on any type of georeferenced data from different resources. Furthermore, several extensions for existing GIS programs are available on the market for specific purposes (engineering applications, optimization, etc.). In general, any extension running inside a GIS application is said to be tightly coupled with GIS. Conversely, stand-alone applications that only process or output GIS data are said to be loosely coupled with GIS.

The aim of this study is to develop a GIS-integrated (with loose coupling) computer software which optimizes the solid waste collection routes. For this purpose, two different route optimization models are generated and resulting routes are compared. The objective of the first model is to minimize the total distance travelled by the trucks. In the second model, the objective is to minimize the fuel consumption, which is a function of distance travelled, the road inclinations and the instantaneous loading conditions of the truck. Information about the real road network is derived through a GIS and integrated into the optimization models. The estimation of fuel consumption for waste collection has been made based on the model used by Tavares et al. (2009).

Bahçelievler and Emek neighbourhoods in Ankara/Turkey are selected as the application area. A digital road network covering the major roads of the entire Ankara and the streets within Bahçelievler and Emek neighbourhoods is used. The road directions are also included in the network. A digital elevation model is used to display the 3D terrain and to calculate the road inclinations.

Once the data are input to the program, they undergo a pre-processing stage so as to prepare the input for the optimization algorithm. The pairwise shortest paths and distances between all the waste collection points are computed, creating a distance matrix. The distance matrix is asymmetrical as a result of unidirectional streets. A single distance matrix would define a single path between any two service points. There are cases where this would end up with a U-Turn on bidirectional roads in the optimization stage. In order to avoid the U-Turns, a secondary shortest path has also been calculated for all nodes located at bidirectional roads. In the calculation of the second shortest path, the path is forced to start from the direction opposite to that in the shortest path. Thereby, two distance matrices are constructed to be used by the optimization algorithm.

The distance matrices and shortest paths between all pairs of waste collection points are the basic requirements of the route optimization algorithm. The problem of finding optimum routes for Solid Waste Collection is a combinatorial optimization problem and is similar to the well-known Vehicle Routing Problem (VRP) with respect to its constraints. According to Machado et al. (2002); exact solution methods are not suitable for large instances of the vehicle routing problem as the search space grows exponentially with increasing problem size. Metaheuristic methods are commonly used for solving routing problems due to their capability to converge into high quality solutions within a reasonable time (Tarantilis et al., 2005). In this study a parallel hybrid genetic algorithm has been implemented to solve the optimization models.

Genetic Algorithms are stochastic optimization techniques that model the natural phenomena of genetic inheritance and Darwinian strive for survival (Michalewicz, 1992). The evolutionary process starts with an initial population of candidate solutions. Each candidate solution is called a chromosome. The performance of each chromosome is evaluated by means of a fitness function to see how well they perform with respect to the objective function (Al Jadaan et al., 2008). A selection procedure is applied to determine which chromosomes are chosen for genetic reproduction (crossover), by which a child chromosome (offspring) is produced from two parent chromosomes. Practically, better fit chromosomes have a higher chance of being selected. The mutation operator makes random modifications on the chromosomes to extend the search space. At each generation, the new population replaces the old one, simulating an evolutionary process.

The advantages of Genetic Algorithms (GAs) for routing problems is that they do not require detailed knowledge of the problem and can easily adapt to changing conditions (Sengoku and Yoshihara, 1998). Moreover, they can provide a solution at any instance of optimization, without the user having to wait for the termination of the algorithm. Genetic Algorithms are among the class of Metaheuristic Optimization Algorithms and are capable of searching globally and converging to nearly-optimal solutions in a relatively short amount of time (Sengoku and Yoshihara, 1998). Furthermore, other hill-climbing methods can easily be implemented within the genetic algorithms to enhance the search capabilities.

The genetic algorithm is parallelized by the course grained PGA islands model (Shengjun et al., 2008). In this model, multiple instances of genetic algorithms run independently on different processors and the solutions produced by each instance migrate periodically from one instance to another. The method suggested by Shengjun et al. (2008) has been implemented by using an exchange pool which enables different instances of genetic algorithms to push and pull the selected routes.

The validation of the implemented optimization algorithm is performed by running the software on 4 asymmetrical TSP benchmark problems (TSPLIB, 2010). The optimum results of the benchmark problems are already known. The results obtained are compared with the best possible solutions. The optimization method has also been tested on the real road network with small test problems involving 10-20 waste collection points.

The output of the optimization algorithm is a set of routes for waste collection vehicles. The routes can be displayed in a 3D terrain view with different colors. The visiting order of the waste collection locations is indicated by numbers. This type of visualization enables the user to view the terrain relief while analyzing the path and to compare the results of different routing models.

In this study, it is aimed to illustrate the difference between the routes generated by two models in 3D view. The first model aims to minimize the total distance travelled whereas the second one aims to minimize the fuel consumption. The visual characteristics of the routes generated are expected to reflect the optimization model used.

Following this introductory chapter, Chapter 2 is dedicated to the literature review of existing studies on Vehicle Routing Problem and solution methods in the literature. Different variations of the Vehicle Routing Problem are explained. A number of different solution strategies to the problem for different cases are given. A brief discussion of the genetic algorithms designed for the vehicle routing problems in the literature is provided. Lastly, existing studies in the literature about route optimization for the solid waste collection case is discussed. Chapter 3 describes the methodology of the study by presenting the framework of the stages involved in the development of the software. After presenting the study area, the data preparation and pre-processing stages are explained. Two new routing models that take into account the path conditions are introduced. The details of the genetic optimization algorithm are given along with various choices and the logic behind them. The parallelization method is explained. Ultimately, the results are discussed. In Chapter 4, detailed information about the implementation of the genetic algorithm is presented. The genetic operators used as well as the tuning of the parameters are explained. The parallelization method and the benchmarking results are given. Chapter 5 describes the area of application and presents the results of the developed software on a number of small test cases and the real case study. A brief discussion of the obtained results is presented. Chapter 6 concludes the main body of the thesis and gives recommendations for the future studies.

Following the main body of the thesis, Appendix A presents a simple manual for the developed software. In Appendix B, fragments of codes and data structures used in the program are presented.

CHAPTER 2

OVERVIEW OF ROUTE OPTIMIZATION AND SOLID WASTE COLLECTION

Route optimization is a broad term covering all attempts to find the shortest routes that cover all the locations that need to be serviced. A number of different route optimization problems are studied in the literature due to their high applicability to many real-life situations such as solid waste collection, milk float routing, mail delivery, school bus routing, heating oil distribution, parcel pick-up and delivery, dial-a-ride systems, etc. (Rizzoli et al., 2004).

This chapter is dedicated to the literature survey of the previous studies. First, the vehicle routing problem is explained briefly, covering different models and general solution methods in the literature. As the proposed optimization method uses genetic algorithms, the next section is devoted to genetic algorithms applied for route optimization. Lastly, the studies related with solid waste collection problem are discussed.

2.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) was first introduced by Dantzig and Ramser in 1959 and has been studied for many different real-life cases ever since. It is a complex combinatorial optimization problem in which the objective is to minimize the total distance travelled by a set of vehicles while servicing all locations. VRP has different models based on the constraints of the real-life situation. The variations and different models of the problem are well documented in Toth and Vigo (2000) and Rizzoli et al. (2004).

The Capacitated VRP is the most basic variant of VRP, in which, a fleet of vehicles of uniform capacity are required to service a fixed number of demand points. Vehicles have a limited capacity and therefore, each vehicle is supposed to go back to the depot once they reach the capacity limit. According to Rizzoli et al. (2004), removing the capacity constraint and limiting the number of vehicles to one, the problem reduces to the classical Traveling Salesman Problem (TSP). Therefore, it can easily be proved that the CVRP and other variants of VRP are NP-Hard in complexity.

The VRP with time window constraints (Rizzoli et al, 2004; Tan et al, 2001) (VRPTW) is especially important in logistics management, in which the company is expected to deliver a number of items from depot(s) to a number of customers on time (i.e., some or all of the items have to be delivered within a time frame). In VRPTW, each customer is associated with a time interval and is required to service each customer within the time window. Tan et al. (2001) addresses some other real-life applications of VRP with time window constraints such as school bus routing, mail and newspaper delivery, fuel oil delivery and municipal solid waste collection; proposing a hybrid solution involves the cooperation of different artificial intelligence techniques such as simulated annealing, tabu search and genetic algorithm. The paper further claims that efficient routing and scheduling of vehicles can save millions of dollars for governments and industries.

In VRP with Pickup and Delivery (VRPPD), the vehicles are supposed to satisfy a set of transportation requests. A transportation request involves transferring the demand from pick-up point to the delivery point. In this problem, the transport items are not originally concentrated in the depots, but they are distributed over the nodes of the road network. In case the demands to be transported are people, the problem usually includes time window constraints in order to prevent customer from waiting too long (Rizzoli et al., 2004). VRPPD arises in a wide range of commercial service companies. A major case for the application of VRPPD is the grocery industry, in which the supermarkets serve as delivery points and their suppliers such as poultry processors or vegetable and fruit wholesalers are pick-up points (Mosheiov, 2008).

Time dependent VRP is an extension of VRPTW, in which the costs between the delivery points depend on time. This situation practically occurs in most cities since the time required to travel from one point to another depends on traffic load which depends on the time of the day (Rizzoli et al., 2004). The speed distributions have to be known before the optimization starts in order to enable the system to calculate the travel times. This variant of the VRP is motivated by the fact that the role of traffic conditions cannot be underestimated in urban areas in order to perform a feasible and realistic optimization (Donati et al., 2008).

Multiple Depot VRP (MDVRP) was formulated by Sumichrast and Markham(1995) and differs from the classical capacitated VRP in that there is more than one depot in MDVRP and each demand point is associated with a depot (Ho et al., 2008). As there is more than one depot involved, the decision makers have to determine which customers are served by which depots

2.2 Solution Methods

The solution methods for the VRP can be broadly classified as exact, heuristic and metaheuristic. Exact methods are those which explore the entire search space for the problem to find the best possible solution to the VRP instance. Exact approaches to solving VRP require algorithms that generate lower and upper bounds to the optimal value of the cost. In general, an upper bound to the optimal value of the problem instance can be obtained by utilizing any heuristic method that can find a feasible solution. In VRP, any set of tours that cover all the demand points constitute a feasible solution with a given cost which cannot be smaller than that of the minimum cost tour. A lower bound to the optimal value of the cost can be determined by solving a relaxation of the optimization problem. A relaxation is another optimization problem whose feasible solutions encompass all feasible solutions of the original problem and whose objective function value is less than or equal to that of the original problem. The optimality of the solution is guaranteed when the lower and upper bounds coincide. The two exact methods that were studied extensively:

- Branch and Bound (Fisher, 1994)
- Branch and Cut (Toth and Vigo, 2002).

The heuristic methods for optimization problems perform a relatively limited exploration on the search space. The aim of heuristics is to produce relatively good solutions as quickly as possible (Tarantilis, 2005). The heuristic methods are broadly classified into three categories:

- Constructive Heuristics
- Local Search Improvement Heuristics
- Two-phase Algorithms

Constructive heuristics use the data of the problem to build a solution gradually (one point at a time) without an improvement phase. Typically no solution is obtained until the procedure is complete.

Local search improvement heuristics are iterative search procedures that gradually improve the solution quality starting from an initial feasible solution by applying a series of local modifications. The initial feasible solution is usually output of a constructive heuristic (Tarantilis et al., 2005).

The two-phase algorithms are classified into two broad categories: “cluster first route second” and “route first cluster second”. The former class of algorithms first clusters the demand points into feasible routes and then constructs the actual routes using feedback loops between the two stages. The examples to this approach are the sweep algorithm (Gillet and Miller, 1974), Fisher and Jaikumar Algorithm (1981) and Petal Algorithm (Ryan et al., 1993). The latter class of algorithms initially find a large single route by utilizing a TSP algorithm disregarding the side constraints and then decompose the tour into feasible vehicle routes in the second phase. Table 2.1 explains a number of heuristic methods in the literature.

Table 2.1 – Heuristic methods

Type	Method
- Constructive Heuristics • Clark and Wright Savings Algorithm (Clark and Wright, 1964; Altinkemer and Gavish, 1991)	Gradually construct the solution, one step at a time without an improvement phase
- Local Search Improvement Heuristics • Cyclic Transfer Algorithm (Thompson and Psaraftis, 1993) • 2-cyclic exchanges (Van Breedam, 1994; Kinderwater and Savelsbergh, 1997)	Iteratively improve the solution starting from an initial feasible solution
-2-Phase Algorithms • Cluster First, Route Second (Gillet and Miller, 1974; Fisher and Jaikumar, 1981; Ryan et al., 1983) • Route First, Cluster Second	Separating the clustering and routing processes, thereby reducing the VRP into several TSP problems.

Metaheuristic methods are higher level heuristic procedures that are designed to guide heuristic approaches on achieving better quality solutions for difficult combinatorial optimization problems (Tarantilis et al., 2005). Metaheuristic Methods perform a deeper exploration of the solution space. Therefore, the quality of solutions produced by metaheuristic algorithms is much higher than that of classical heuristic methods (Tarantilis et al., 2005). Each metaheuristic has one or more adjustable parameters. Although this provides flexibility, tuning the parameters requires careful calibration and testing on an independent set of problem instances (Tarantilis et al., 2005). The metaheuristic algorithms employ techniques that provide both intensification and diversification of the candidate solutions. Intensification refers to the exploitation of the existing solutions to obtain better local-optimum solutions. Diversification, on the other hand, refers to the exploration of the entire search space so that the algorithm does not get stuck in a low-quality locally optimal solution. Following is a list of metaheuristic algorithms that are commonly implemented for routing problems:

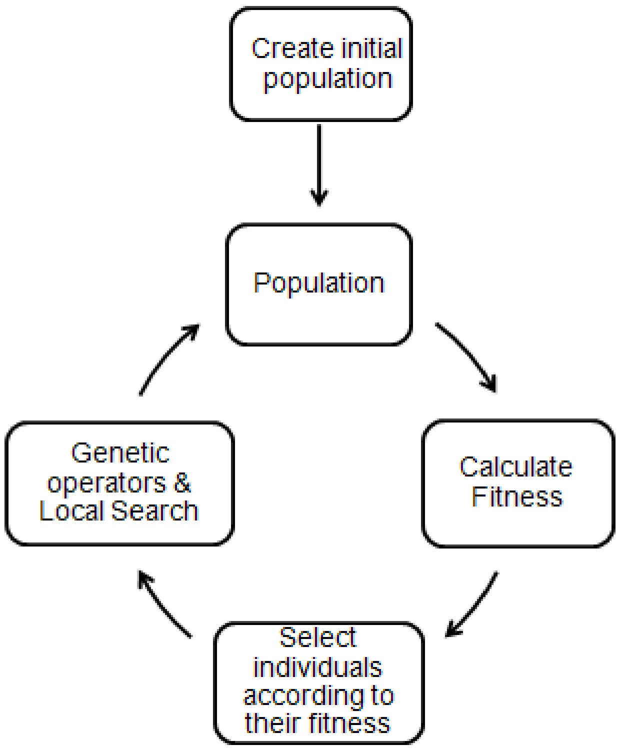
- Simulated Annealing
- Threshold Accepting Metaheuristic
- Ant Colony Optimization
- Genetic Algorithms

2.3 Genetic Algorithms for Route Optimization

The search methods of Genetic Algorithms model the natural evolutionary processes to gradually improve the feasible initial solutions. It is inspired by the Theory of Natural Selection proposed by the British naturalist Charles Darwin in 1859. The theory states that the individuals possessing favourable characteristics are more likely to survive and reproduce, which makes them more likely to transfer their genetic material to the proceeding generations. The individuals with less favourable characteristics will gradually diminish as they are less likely to survive although some individuals manage to do so by luck. As the organisms mate, the genetic information of the parents is transferred to the offspring.

In Genetic Algorithms (GA), each candidate solution to the problem is referred to as a chromosome. In the routing problem case, a chromosome refers to a complete set of routes that utilize all vehicles and cover all the points that need to be serviced. The algorithm starts with an initial population of chromosomes.

The initial chromosomes comprising the initial population are constructed either randomly or by using heuristic methods. After an initial population of feasible solutions is generated, the fitness of each individual in the population is calculated. The fitness of a chromosome is a measure of how the chromosome fits with the objective function. A selection procedure is used to decide which chromosomes undergo evolutionary processes using genetic operators. Generally, better fit chromosomes have a higher chance of being selected. The crossover operator combines the features of two solutions by swapping genes in order to form offsprings that carry the genetic properties of parents (Michalewicz, 1992). The assumption in using crossover operator is that the genetic combination of two good solutions yields another good solution having common properties with the parents. The mutation operator makes random or heuristic modifications on the selected chromosomes, creating variations in the population. Mutation helps the optimization algorithm to escape from local-minima traps and explore a wider range of the solution space. During the evolutionary process, hill-climbing methods can be used to perform local search on selected individuals. The hill climbing methods test the neighbour genes of a chromosome for different combinations and if a modification results in a better value of objective function, that modification is accepted. This process continues until no further improvement is possible. At the end of each iteration, a new generation is created from the previous one. Figure 2.1 illustrates the general genetic algorithm cycle.



Create initial population
Population
Calculate Fitness
Select individuals according to their fitness
Genetic operators & Local Search

Figure 2.1 – Generalized Genetic Algorithm Cycle

2.3.1 Representation of Path

For combinatorial optimization problems such as Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) in which a permutation of points is of interest, binary representation of tours is not well suited as the genetic operators cannot be applied in a meaningful way. This is mostly because modification of a single bit may result in an illegal tour where a repair algorithm is required in order to fix the chromosome (Michalewicz, 1992). Furthermore, it would be so difficult to handle the genetic crossover operator which is supposed to produce new routes inheriting the genetic properties of parent (Michalewicz, 1992).

The most common representation of chromosomes in the literature is the vector representation. Vector representation is commonly implemented as an array of numbers each corresponding to a demand point. Michalewicz (1992) describes three different vector representations (Table 2.2) existing in literature in connection with TSP:

- Adjacency Representation
- Ordinal representation
- Path Representation

Table 2.2 – Comparison of Vector Representation Methods of Route Paths

Representation	**Explanation**	**Comments**
Adjacency	If there is a direct tour between city i and j, city j is listed in position i.	Illegal routes can be produced, which requires repair algorithms.
Ordinal	For each point in the route path R, the index of the point within the reference list is recorded and inserted into the ordinal representation vector O. The point is then deleted from the reference list.	Hard to implement the genetic operators.
Path	The route path is represented directly.	The most natural representation. Preferred due to its simplicity and applicability to various operators.

2.3.2 Generation of the Initial Population

The construction of initial population is of great importance since it has great impact on the genetic material comprising the final solution (Bjarnadóttir, 2004). The initial population is generally constructed randomly; however it can also be constructed using heuristic methods.

Many classical and hybrid genetic algorithms prefer random generation of initial routes (e.g., Sengoku and Yoshihara, 1998; Nazif and Lee, 2010). As its name implies, all the connections in randomly generated routes are established by randomly selecting a point from the available point list. The randomly generated routes possess a high diversity of genetic material which enables the genetic algorithm to explore a larger search space. However, it generally takes a considerable amount of computation time to obtain optimized solutions.

The heuristic approaches aim to construct well-structured routes. One of the possible heuristic approaches is to use the sweep approach of Gillet and Miller (1974) in which the demand points are sorted according to polar angle from the depot. The second approach is to solve the Generalized Assignment Problem to obtain an allocation of demand points for vehicles and then to construct the routes by using simple heuristic methods. A third heuristic approach proposed by Ho et al. (2008) involves grouping the demand points by assigning them to nearest depot (in case of multiple depots), generating routes using the Clark and Wright (1964) saving algorithm and then solving a scheduling problem. The application of heuristics methods in the generation of initial population is expected to result in solutions that evolve faster. However, a possible drawback is that such an artificially constructed population may lack the diversity required to obtain near-optimal solutions (Baker and Ayechew, 2003).

2.3.3 Calculation of Fitness

Each individual in the population is a potential solution to the problem and is evaluated by means of a measure called “fitness”. Fitness can be considered as a way of ranking a chromosome against other chromosomes. Therefore, the fitness function can be associated with the objective function of the model. The raw fitness value in routing problems is generally expressed as the inverse of the cost (Michalewicz, 1992). The normalized fitness, on the hand is calculated such that the sum of all normalized fitness values add up to 1.

2.3.4 Selection Operator

Selection operator chooses the chromosomes to be included in the next generations. The chromosomes may undergo genetic operations such as crossover and mutation before transferring to next generations. The selection operators are generally stochastic and are formulated in such a way that the better members of the population have a greater probability of being selected. It is important that worse members of the population have some probability of being selected as well (Al Jadaan et al., 2008). Three selection operators are explained:

- Tournament Selection
- Ranking Selection
- Roulette Wheel Selection.

Tournament selection involves choosing a sample of k individuals from the population randomly and running several tournaments among the chosen individuals. The best individual in the tournament is selected within a probability of p, which is also called as the pressure of selection (Miller and Goldberg, 1995). The probability of selecting nth best individual in the tournament is given as:

$$P_s = p \times (1 - p)^n \tag{2.1}$$

Ranking selection (Miller and Goldberg, 1995) sorts the individuals in the population based on fitness. Assuming that there are N individuals in the population, the probability of being selected for an individual in the sorted list is:

$$P_s = \frac{2 \times Rank}{N(N + 1)} \tag{2.2}$$

In Roulette Wheel Selection (RWS), the selection probability of each individual is proportional to its fitness (Miller and Goldberg, 1995). In practice, the selection probability of an individual equals to the normalized fitness of the individual since the normalized fitness values add up to 1.

$$P_s = \frac{f_i}{\sum_{j=1}^{n} f_j} = fn_i \qquad (2.3)$$

2.3.5 Crossover Operator

The crossover operator creates new chromosomes using the genes of the parent chromosomes. The parent chromosome pairs to be mated are selected by means of the selection operator. The performance of the crossover operator is directly related with its ability to transfer the significant properties of the parent chromosomes to the child. The following crossover operators are commonly used for routing problems:

- Partially Mapped Crossover (PMX)
- Order Crossover (OX)
- Edge Recombination Crossover (ERX)

The Partially Mapped Crossover (PMX), introduced by Goldberg and Lingle (1985), builds an offspring by choosing a subsequence of tour from one parent and preserving the order and position of as many cities as possible (Michalewicz, 1992). In PMX, two random cut points for the parent chromosomes are selected to serve as a boundary for swapping operations. For example, let p_1 and p_2 be two chromosomes with marked cut points such that:

$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)\ and$

$p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3).$

Swapping the segments bounded by the cut points, the offsprings will initially be:

$o_1 = (x\ x\ x \mid 1\ 8\ 7\ 6 \mid x\ x)\ and$

$o_2 = (x\ x\ x \mid 4\ 5\ 6\ 7 \mid x\ x).$

The swap operation also defines a series of mappings between points $(1 \leftrightarrow 4, 8 \leftrightarrow 5, 7 \leftrightarrow 6, 6 \leftrightarrow 7)$ which assist in completing the remaining parts of the offspring from original parents. If a point to be copied from an original parent conflicts with the swapped segment (i.e., results in visiting the same point twice), the mapping of that point is used instead, which guarantees the avoidance of any conflict. The resultant offsprings are:

$o_1 = (4\ 2\ 3\ |\ 1\ 8\ 7\ 6\ |\ 5\ 9)\ and$

$o_2 = (1\ 8\ 2\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3).$

Order Crossover (OX), introduced by Davis (1985), builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of demand points from another parent (Michalewicz, 1992). In OX, first the segments between cut points are copied into the offsprings. Considering the same parents used in PMX, the offsprings after the first step would be:

$o_1 = (x\ x\ x\ |\ 4\ 5\ 6\ 7\ |\ x\ x)\ and$

$o_2 = (x\ x\ x\ |\ 1\ 8\ 7\ 6\ |\ x\ x).$

Next, starting from the second cut point of one parent, the points are copied from other parent excluding the points already used. For instance, the sequence of points in the second parent is $(9\ 3\ 4\ 5\ 2\ 1\ 8\ 7\ 6)$. After the removal of points already used, the sequence becomes $(9\ 3\ 2\ 1\ 8)$. Appending this sequence to the offspring starting from the second cut, the offspring becomes:

$o_1 = (2\ 1\ 8\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3).$

Similarly, the second offspring becomes:

$o_2 = (3\ 4\ 5\ |\ 1\ 8\ 7\ 6\ |\ 9\ 2).$

According to Michalewicz (1992), the OX makes use of the fact that the relative orders of the points are important rather than their positions in the path.

The Edge Recombination Crossover (ERX), introduced by Whitley, Starkweather and Fuquay (1989), transfers more than 95% of the edge information from parents to offspring. The idea behind ERX is that an offspring should be constructed based on the edges from both parents rather than considering the position or relative order of the demand points (Michalewicz, 1992). For the two parent chromosome instances given for PMX, the edge list in Table 2.3 can be constructed. Starting from an initial point for ERX, the next point is selected by making use of the edge list. Generally, the point with smallest number of edges in the edge list is selected. If there is more than one alternative, a random choice is made. Repeating the procedure, the following offspring can be generated:

$o_1 = (1\ 4\ 5\ 6\ 7\ 8\ 2\ 3\ 9)$.

Table 2.3 - The Edge List for Edge Recombination Crossover (ERX) Example

Point No	Edges to other points
1	9 2 4
2	1 3 8
3	2 4 9 5
4	3 5 1
5	4 6 3
6	5 7 9
7	6 8
8	7 9 2
9	8 1 6 3

2.3.6 Mutation Operator

The mutation operator makes random changes to chromosomes to maintain the diversity within the population. Thereby, premature convergence to local optimum solutions is prevented. Three mutation algorithms are explained:

- Swap Sequence Mutation
- Inversion Mutation
- Heuristic Mutation

The swap sequence operator (Nazif and Lee, 2010) randomly selects two substrings of demand points and exchange them. Although swapping sequences is one of the simplest mutation techniques available in the literature, it is very useful for the exploration of the solution space. Figure 2.2 illustrates the swap sequence mutation technique.

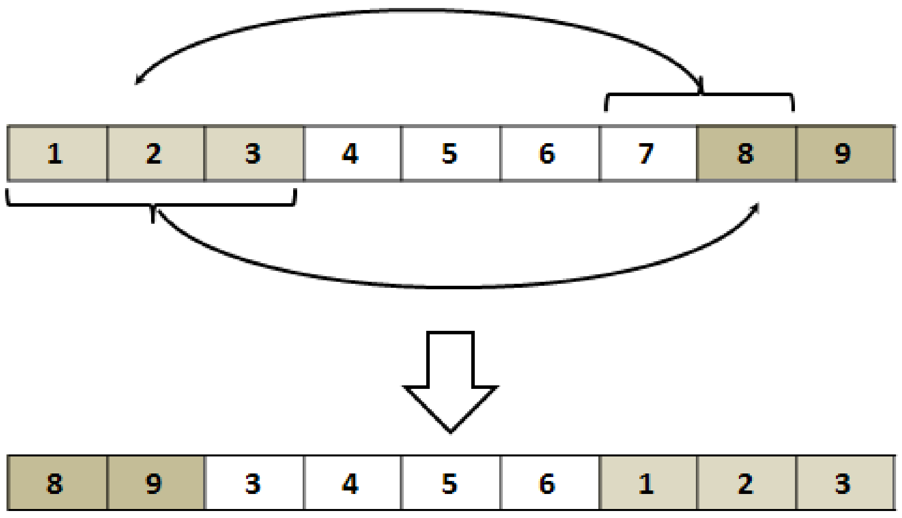


Figure 2.2 - Swap Sequence Mutation

The inversion mutation, on the other hand, selects a substring from the parent chromosome and flips it to form a new offspring (Ho et al., 2008). The inversion operator provides the system with the capability to find the suitable ordering of demand points. Figure 2.3 illustrates the inversion mutation.

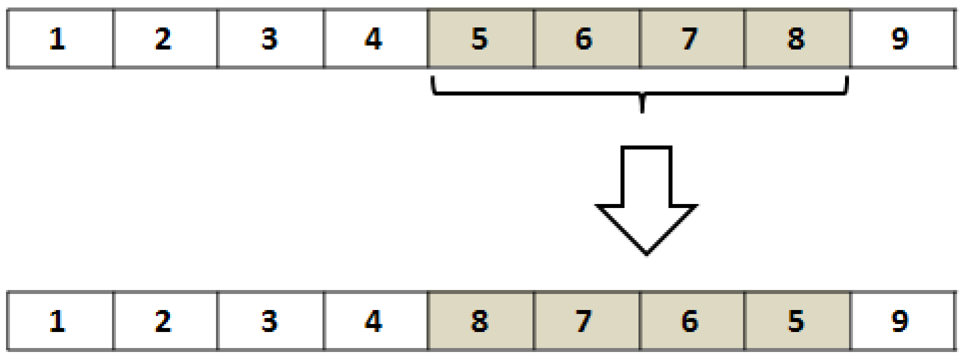


Figure 2.3 - Inversion Mutation

In general, the heuristic mutation operators use a neighbourhood technique to produce better offsprings. A set of chromosomes transformed from the parent chromosome by exchanging genes are regarded as the neighbourhood (Ho et al., 2008). Generally, best chromosome in the neighbourhood is used as the offspring.

The 2-opt mutation (Sengoku and Yoshihara, 1998) is one of the most commonly used heuristic mutations which improves the tours by checking every pair of edges and checks if exchanging the pairs result in an improvement. If there is any improvement in case of an exchange, the pairs are swapped and the order of the subtour is reversed. Figure illustrates the 2-opt mutation algorithm.

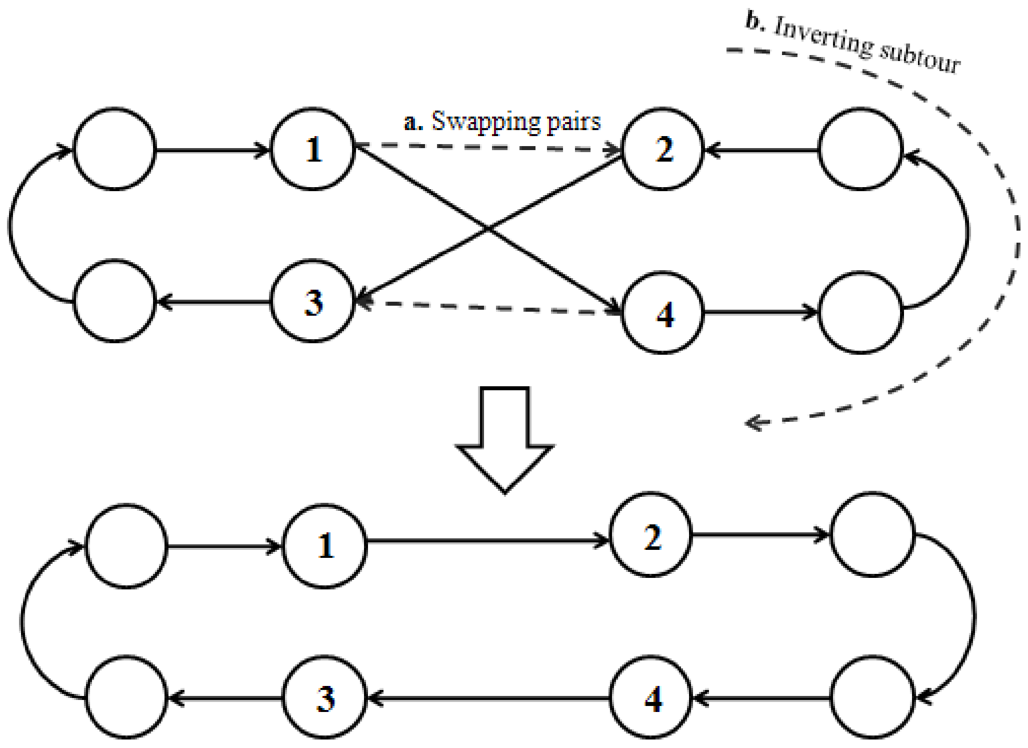


Figure 2.4 - 2-opt Heuristic Mutation

2.4 Solid Waste Collection

According to Nuortio et al. (2006), planning the solid waste collection is one of the most difficult operational problems encountered by local authorities in solid waste management. Therefore, it is highly important to consider the cost-effectiveness of a solid waste collection operation while designing the collection routes. The waste collection routes in many cities are still designed manually. However, there have been numerous technological advances in the past decades in terms of computational power; which leads the haulers to seriously consider using computer software for planning their routes (Nuortio et al., 2006). Most of the studies involving route planning for the collection of municipal solid waste in the existing literature are based on the Vehicle Routing Problem.

Uraz (2002) used GIS tools to make a descriptive analysis of Bahçelievler Neighbourhood and presented a comprehensive overview of the neighbourhood and the collection service in Bahçelievler. The collection routes were generated using the Network Analyst Extension of ArcView 3.2. It uses Dijkstra's shortest path algorithm to calculate the shortest path from a root node to every other node in the network. At each iteration, the edges next to the node are evaluated regarding their costs and the edge with least cost is selected. According to Uraz (2002), "The Analyst may sometimes generate paths that are not suitable for solid waste collection truck to follow in the streets of the study area. The most important factor of these non-suitable paths is that the analyst allows the vehicle to turn back easily on the same street for reaching to the next node". The cost parameters used in the study are distance, time taken to travel along the streets. The effect of average speed on the fuel consumption was considered as well. A number of different routes were proposed for different zones of Bahçelievler.

Ghose et al. (2006) proposed a solid waste collection system for the municipality of Asansol in West Bengal, India. A GIS-based optimal routing model was considered for transporting solid wastes to the landfill, involving the planning of bins, vehicles and optimal routing. For the collection phase, the vehicles are divided into three categories based on their volume and suitability for different conditions. The vehicle type to be used depends on the road with and the type of the collection bin. The study aimed to propose a framework for efficient solid waste management system rather than concentrating solely on the route optimization. The collection routes were generated by using the network module of Arc/Info GIS software with objective of finding the shortest or minimum impedance through a network. The routes were computed using a travelling salesman problem model.

Nuortio et al. (2006) conducted a study in two different regions of Eastern Finland and proposed a conceptual model aiming to schedule the waste collection activities and minimize the distance travelled by garbage trucks using metaheuristic methods with local search techniques. Detailed historical data provided by the waste management company on waste accumulation levels and stop points of vehicles for waste collection was used to estimate the waste production of each bin and the travelling speed in each road class. Time windows were assigned to each bin, considering the time limitations of the collection activity and working hours of the waste disposal site. The distances between pairs of bins were calculated by using the shortest path algorithm of Dijkstra (Dijkstra, 1959). The travel times were estimated by using historical GPS data. In the optimization stage, a feasible solution was generated by a hybrid insertion heuristic and then a metaheuristic algorithm is used along with local search techniques.

Tavares et al. (2009) proposed a GIS integrated route modelling software to optimize fuel consumption of waste collection trucks in the city of Praia and Santiago Island. The study addresses road slope when estimating the fuel consumption of the waste collection trucks. It was shown that longer routes can be more optimal in terms of fuel consumption when the road inclination is taken into account. In order to estimate the fuel consumption, the model proposed by Ntziachristos and Samaras (2000) was used. Although the effect of vehicle load was also considered in the paper, it was not implemented in the study. The Network Analyst software was used for the optimization of the waste collection vehicle route for a single vehicle. The results showed that road slopes affect the optimal route significantly when the fuel consumption is to be minimized.

This study proposes a more realistic approach by incorporating the spatial constraints for one-way streets and U-Turns into the model and using a digital elevation model. The route optimization is carried out by a hybrid genetic algorithm, which is easy to adapt to new constraints that may arise under varying conditions. This provides additional flexibility compared to exact algorithms and enables the user to work on larger data sets. The search capability of the genetic algorithm is strengthened by using hill-climbing algorithms, which speeds up the process of converging into high quality solutions. Furthermore, the algorithm is parallelized with very high efficiency so as to speed-up the genetic search.

CHAPTER 3

METHODOLOGY

Classical vehicle routing models do not meet the requirements of solid waste collection vehicle routing as far as the path constraints are considered. One of the important path constraints is the avoidance of U-Turns (Vesilind et al., 2001). Therefore, a spatial routing model that includes vector representation of routes needs to be established and solved. Moreover, factors affecting the fuel consumption such as slope and instantaneous vehicle load need to be included within the model.

3.1 General Framework of the Methodology

The aim of the proposed framework is to optimize the waste collection routes using two different spatial models. The objective of one of the models is to minimize the total distance travelled whereas the other one aims to minimize the fuel consumption. The framework of this study consists of four main stages: input, pre-processing, optimization of waste collection routes and display of output. Figure 3.1 illustrates the general framework of this study.

The input stage involves the preparation of the input data of the application. The road network and waste collection locations are digitized and exported as an ESRI Shape File (ESRI Shape File Technical Documentation, 2010). The road directions are indicated as an attribute. As a result, uni-directional and bi-directional roads can be distinguished during route optimization.

In the pre-processing stage, road network and waste collection points are used to generate the distance matrices in which all the shortest distances between waste collection points are declared. For this purpose, firstly, the projections of the waste collection points on the nearest street features are calculated. Then, a directed graph is created from the road network and the nodes calculated. By using the graph, the shortest paths between all pairs of waste collection points are calculated. For the points lying along the bidirectional routes, second shortest paths that are forced to start from the direction opposite to the primary shortest path are calculated as well. For each path generated, the average slope of the path has also been calculated for usage in the estimation of the fuel consumption.

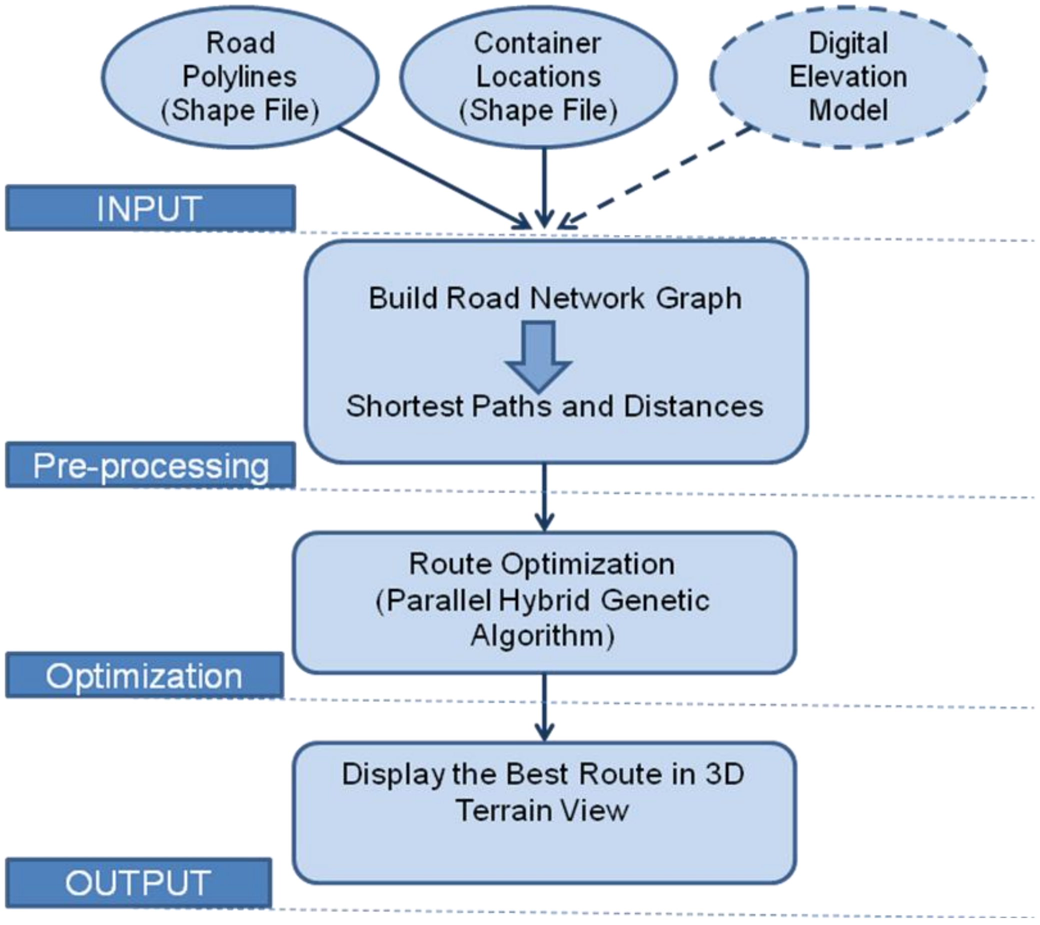

Road Polylines (Shape File)
Container Locations (Shape File)
Digital Elevation Model
INPUT
Build Road Network Graph
Shortest Paths and Distances
Pre-processing
Route Optimization (Parallel Hybrid Genetic Algorithm)
Optimization
Display the Best Route in 3D Terrain View
OUTPUT

Figure 3.1 - Framework of the Study

In the optimization stage, a parallel hybrid genetic algorithm is used to search for the routes with lowest total cost that service all the required demand points (waste collection locations). In this context, the term “cost” depends on the objective of the optimization. Two optimization models are proposed. The first model aims to minimize the total distance travelled by all vehicles and the cost refers to distance in this model. On the contrary, the second model aims to minimize the total fuel consumption. In this case, cost refers to fuel consumption.

The genetic optimization algorithm starts by creating an initial population of feasible solutions. Then, an iterative evolutionary process begins, in which selected good solutions combine with each other to produce other solutions (offspring) that inherit the properties of their parents and some solutions are exposed to modifications. This evolutionary process is supported by hill-climbing algorithms that periodically make local improvements on the solutions. The genetic operators are designed in such a way that no invalid routes are produced at any instance of execution. This is ensured by keeping track of the assignment status of each waste collection points in every genetic operator. The algorithm is parallelized by using islands model (Shengjun et al., 2008). Thereby, multiple instances of the optimization algorithm run at the same time and communicate periodically. The genetic algorithm is capable of providing a solution any time without having to wait for the algorithm to terminate.

The last stage (output) involves displaying the resultant routes in an illustrative manner. The optimization algorithm gives the arrangement of waste collection points to the output module. The waste collection points are represented by an array of numbers indicating the waste collection point and whether the primary or secondary shortest paths are used. The output module sequentially reads the route arrays displays the paths connecting the waste collection locations on a 3D terrain view.

3.2 Input Data

The developed software has three main inputs: a polyline shape file representing road network, a point shape file representing the garbage containers and a digital elevation model represented by a grid file generated by Surfer (Surfer, 2010). The organization of the input files is described in Appendix A.

Table 3.1 shows the expected attributes of the road features. The ID1 field is the primary key, which is unique for each road feature. The direction attribute indicates whether the road is unidirectional or bidirectional. In practice, the traffic flow is either in the direction of digitization or both ways. The road type attribute refers to the road hierarchy which affects the line width of the feature while displaying. The level attribute corresponds to the height level of the feature. This feature is implemented to clarify which roads that share a common latitude and longitude actually intersect. The features having a level value difference greater than 2 are assumed to be not intersecting.

Table 3.1 - The attributes of the road features

Field Name	**Type**	**Explanation**
ID1	Integer	The primary key of the road features
Name	String	The name of the street
Direction	Integer	Direction of traffic flow with respect to digitization direction. (0=same, 1=opposite, 2=both ways)
Type	Integer	The road type. (1=major road, 2=main road, 3=street)
Level	Integer	The height level of the road.

Table 3.2 shows the attributes of point features. The ID1 field is the primary key, which is unique for each point feature. The Type field indicates whether the point is a waste collection point or landfill.

Table 3.2 – The attributes of the point features

Field Name	**Type**	**Explanation**
ID1	Integer	The primary key of the point features
Type	Integer	Type of the point (0=waste collection point, 1=Landfill)

3.3 Pre-processing

The aim of the pre-processing stage is to find the shortest paths and distances between each pair of containers so as to construct the distance matrix required by the optimization algorithm. The calculation of pairwise shortest paths requires a preprocessing on the given input data. The preprocessing stage involves the construction of the graph structure to be used by the shortest path algorithm.

Firstly, the geometric projections of waste collection points on the closest street features are determined. Thereby, each waste collection location is associated with a point on the road network. Each point of projection on the road network is recorded as a marked node. Then, all the intersections within the road network (road-road intersections) are determined and recorded as normal nodes (Figure 3.2).

Each node in the network is recorded with its position within the containing polyline. The nodes generated and polyline are used to construct the graph structure consisting of vertices and edges, denoted by $G(V, E)$.

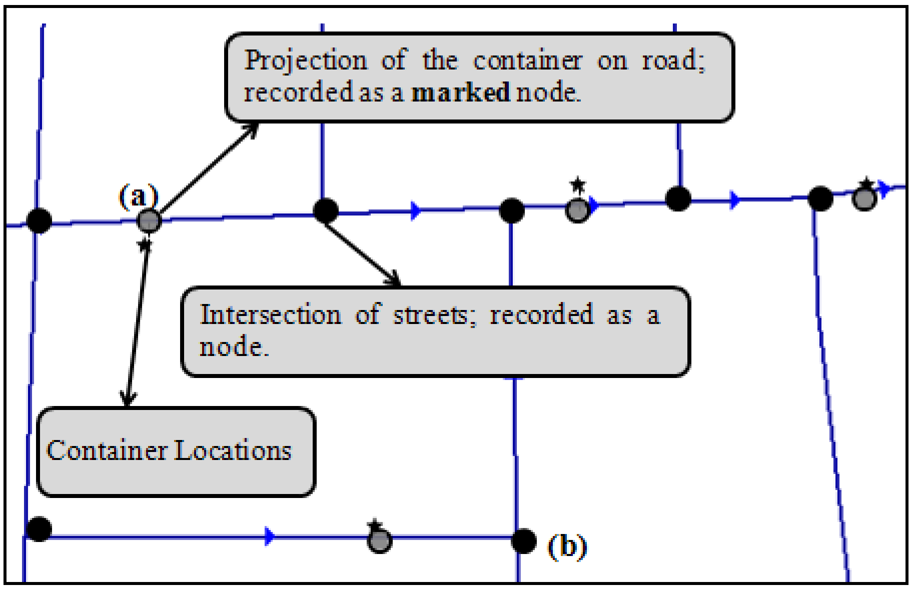

Projection of the container on road; recorded as a **marked** node.
(a)
Intersection of streets; recorded as a node.
Container Locations
(b)

Figure 3.2 - Preprocessing the GIS data. a) projecting the containers to streets, b) calculating the street intersections and marking them as nodes

Once the graph is constructed, all pairs of shortest paths between the “marked nodes” are calculated by means of Dijkstra’s shortest path algorithm. The algorithm is modified so that the nodes closer to the destination point have a higher priority during the search. This modification enables the algorithm to find a short path to the destination earlier. In this manner, longer paths are eliminated without having to search unnecessary path routes.

The edges represented by straight lines in traditional vehicle routing problems are replaced by the shortest paths calculated using the road network. Therefore, the distance between any two containers is represented by the shortest distance calculated within the road network and the edge is represented by the shortest path.

Even though this method provides sufficient input for the optimization algorithm by generating the distance matrix and the edges, it does not address an important path constraint for bidirectional roads: using shortest paths may lead to invalid routes in cases where two consecutive edges have a common node lying along a bidirectional road. It is probable that the ending vector of the former edge and the starting vector of the latter edge head to opposite directions. In these cases, a second shortest path must be used for the second edge to avoid a U-turn on the bidirectional road. The second shortest path is calculated using the same algorithm by forcing the opposite direction for the starting vector of the second edge. This case is illustrated in Figure 3.3.

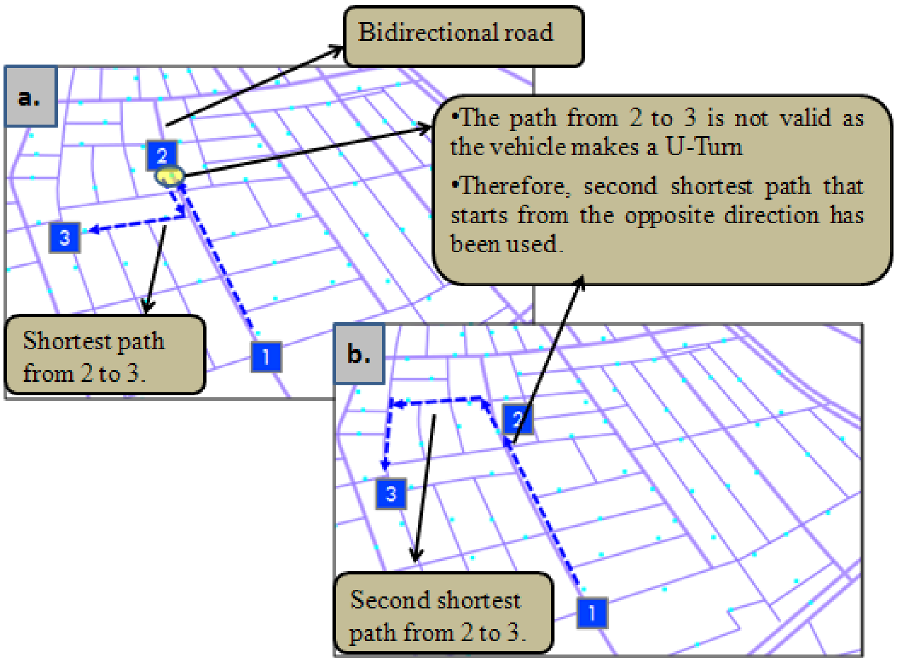

Bidirectional road
a.
•The path from 2 to 3 is not valid as the vehicle makes a U-Turn
•Therefore, second shortest path that starts from the opposite direction has been used.
Shortest path from 2 to 3.
b.
Second shortest path from 2 to 3.
1
2
3

Figure 3.3 - An example invalid route problem. a) An invalid route as the two consecutive edges are in opposite directions (U-Turn), b) Corresponding validated route which uses the second shortest path for the edge between nodes 2 and 3.

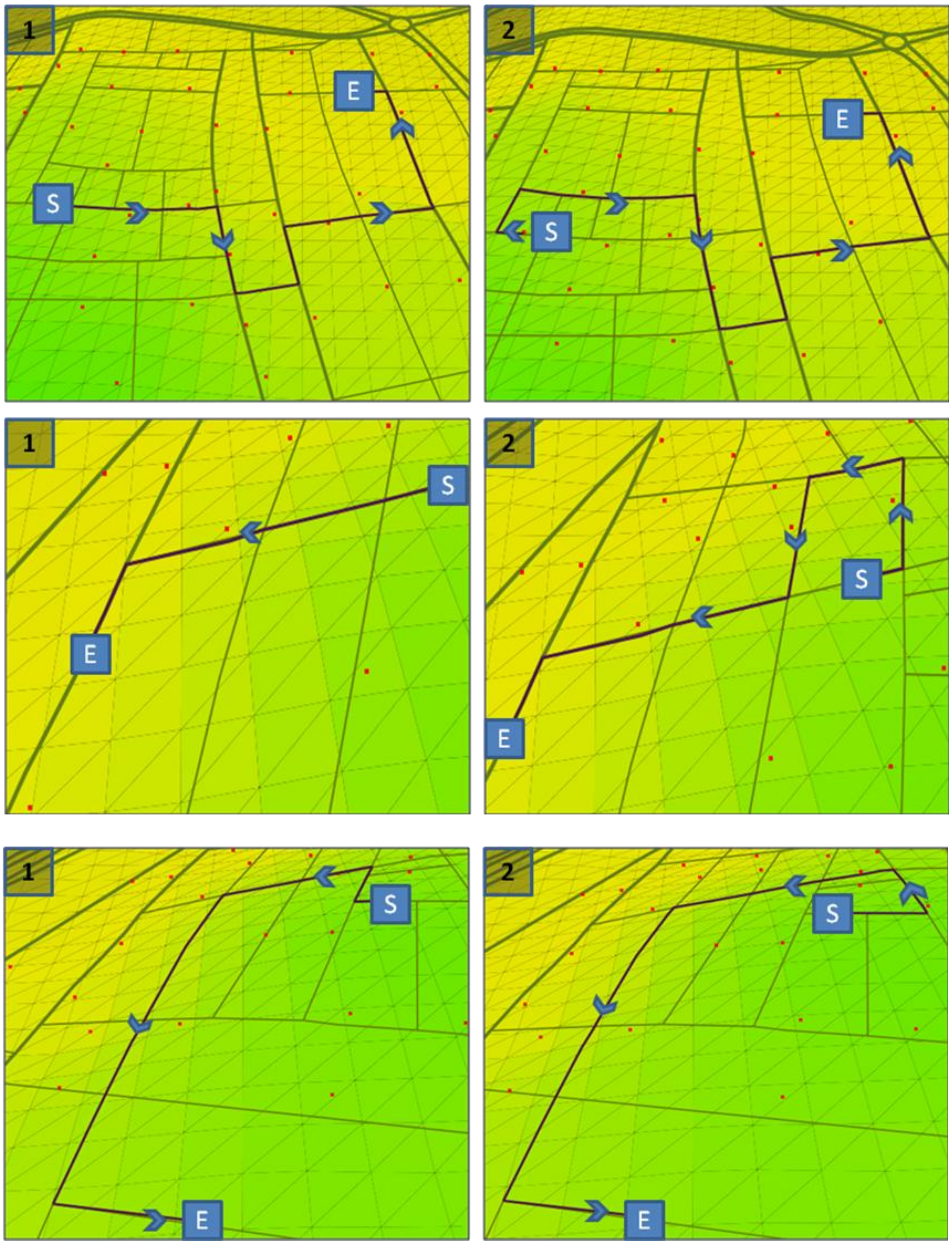


Figure 3.4-First and Second Shortest Paths

The average slope of the paths have also been calculated to be used in the optimization model that takes into account the fuel consumption. The slope of each segment (straight line) comprising the shortest path is calculated from the available altitude data. The average slope of the path is the weighted average of all the slopes comprising the path, in which the affect of the slope is proportional to the length of the line segments.

$$\lambda_{avg} = \frac{\sum_{i=1}^{n} \lambda_i l_i}{\sum_{i=1}^{n} l_i} \qquad (3.1)$$

where

λ_{avg} is the average slope of the path,

n is the number of line segments comprising the shortest path,

λ_i is the slope of the line segment i and

l is the length of line segment i.

3.4 Optimization

The optimization stage aims to generate a route for each vehicle in such a way that each route starts and ends at the landfill and each waste collection point is serviced by exactly one vehicle, minimizing the total operational cost. Two optimization models are proposed. The first one aims to minimize the total distance travelled whereas the second one aims to minimize the fuel consumption; which is a function of distance, instantaneous vehicle load and the road gradient. In both models, the term “cost” is used to address the item to be minimized. Therefore, in the first model, the cost is associated with distance whereas in the second model, the cost is associated with fuel consumption.

3.4.1 Optimization Models

Let there be m vehicles and n waste collection points which are required to be serviced. The route optimization problem for solid waste collection can be defined on a directed graph $G(V,E)$ where $V = \{u_0, u_1, u_2, \dots u_N\}$ is the set of vertices and $E = \{(u_i, u_j): u_i, u_j \epsilon V, i \neq j\}$ is the set of edges defined on the graph G. The vertex u_0 represents the landfill and the other vertices represent the waste collection points.

Let M be the set of vehicles and $R_1, R_2, \dots, R_m$ be the sets of routes assigned to each vehicle. The route of each vehicle k can be shown as set of vertices (3.2) where u_i^k represents the ith point in the route of vehicle k and n_k is the number of waste collection points assigned to route vehicle k. For convenience, u_0^k and $u_{n_k+1}^k$ refer to the landfill for each route k. Therefore, collection trucks leave from the landfill site and return back when they are filled up.

$$R_k = \{u_0^k, u_1^k, u_2^k, \dots, u_{n_k}^k, u_{n_k+1}^k\}, \forall k \in M \tag{3.2}$$

$where\ R_1 \cup R_2 \cup \dots \cup R_m = V.$

The primary and secondary shortest path distances of any two points u_i, u_j are denoted by $d_1(u_i, u_j)$ and $d_2(u_i, u_j)$, respectively. The geometric vectors $\overrightarrow{v_f^1}(u_i, u_j)$ and $\overrightarrow{v_l^1}(u_i, u_j)$ are the first and last vectors of the primary shortest path and $\overrightarrow{v_f^2}(u_i, u_j)$ and $\overrightarrow{v_l^2}(u_i, u_j)$ are the first and last vectors of the secondary shortest path between the vertices u_i, u_j, which were calculated in the pre-processing stage. The angle between any two vectors is computed by the Equation 3.3, which uses the dot product formula.

$$\Phi(\overrightarrow{v_1}, \overrightarrow{v_2}) = \cos^{-1} \frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{|\overrightarrow{v_1}|\ |\overrightarrow{v_2}|} \tag{3.3}$$

A binary variable x_{ij} associated with each edge in the graph denotes if there exists an edge in graph G that connects vertices u_i and u_j. The value of x_{ij} is 0 if there is no direct connection between the vertices u_i and u_j within the routes and 1 if the vertices have a direct connection. Additionally x_{0j} indicates an edge starting from the landfill and x_{i0} indicates an edge ending at the landfill.

Another binary variable y_i^k indicates whether the primary or secondary shortest paths are used connecting the i^{th} and $(i+1)^{th}$ vertices of route k. The value is 1 if the first shortest path is used and 2 if second shortest paths are used. Equation 3.4 says that the path between the i^{th} and $(i+1)^{th}$ vertices of route k is taken from the primary shortest path if the angle between the last vector of the edge $E(u_{i-1}^k, u_i^k)$ and the first vector of the edge $E(u_i^k, u_{i+1}^k)$ is not opposite. Otherwise, the path is taken from the second shortest path. y_i^k is defined recursively (in terms of itself) since the progress of the route at any instance of construction depends on the previous path.

$$y_i^k = \begin{cases} 1 & if\ i = 0\ or\ \Phi\left(\overrightarrow{v_l^{y_{i-1}^k}}(u_{i-1}^k, u_i^k), \overrightarrow{v_f^1}(u_i^k, u_{i+1}^k)\right) < \pi, \\ 2 & if\ otherwise \end{cases} \tag{3.4}$$

where

$\pi \approx 3.14159$ corresponds to the angle between opposite vectors in radians. The condition that the angle between two adjacent edges is equal to π radians is a U-Turn. In case of a U-Turn, the second shortest paths are used.

The distance between i^{th} and $(i+1)^{th}$ vertices for every route $k \in M$ depends on whether the first or second shortest paths are used. Therefore, D_i^k can be defined as:

$$D_i^k = \begin{cases} d_1(u_i^k, u_{i+1}^k), & if y_i^k = 1, \\ d_2(u_i^k, u_{i+1}^k), & if y_i^k = 2 \end{cases} \tag{3.5}$$

In the first optimization model, the aim is to minimize the total distance travelled. Therefore, the cost of the route is defined in terms of distance. The distance-based cost between i^{th} and $(i+1)^{th}$ vertices for every route $k \in M$ can be defined as:

$$C_{D_i}^{\ k} = D_i^k \tag{3.6}$$

The total cost of route k, therefore, is

$$c_D(R_k) = \sum_{i=1}^{n_k} C_{D_i}^{\ k} \tag{3.7}$$

A solution of the problem consists of a partition $R_1, R_2, \dots, R_m$ of V and a corresponding permutation which specifies the order of the customers in the route k (Tarantilis et al., 2005).

The objective of the model is to

$$Minimize \sum_{k=1}^{m} c_D(R_k) \tag{3.8}$$

Subject to

$$R_i \cap R_j \in \{u_0\}, \qquad for\ \forall i,j \in M, i \neq j \tag{3.9}$$

$$\sum_{k=1}^{m} n_k = n \tag{3.10}$$

$$\sum_{i \in N} x_{i0} = \sum_{j \in N} x_{0j} = m \tag{3.11}$$

$$\sum_{i \in N} x_{ir} = \sum_{j \in N} x_{rj} = 1, for\ \forall r \in N \tag{3.12}$$

$$x_{ij} \in \{0,1\}, \forall i,j \in N \tag{3.13}$$

$$y_i^k \in \{1,2\}, \forall k \in M, i \in \{1,2,\ldots,n_k\} \tag{3.14}$$

where

R_k is the route vector for vehicle k, consisting of vertices,

n_k is the number of vertices in route k,

x_{ij} indicates whether there is an edge between vertices i and j,

y_i^k indicates whether the first or second shortest path is used in the i^{th} edge of k^{th} route.

The objective function, Equation 3.8 is to minimize the total distances travelled. Equation 3.9 restricts the routes so that no two vehicles can service the same waste collection location. Equation 3.10 ensures that the sum of total number of waste collection points assigned to each vehicle adds up to the number of waste collection points. Equation 3.11 says that the landfill is entered and left as many times as the number of vehicles. Equation 3.12 shows that each waste collection point is entered and left only once. Equation 3.13 says that there either exist or not exist an edge between points i and j by restricting the possible values of the variable to 0 and 1. Equation 3.14 restricts the paths that can be used to primary and secondary shortest paths.

In the second optimization model, the objective is to minimize the fuel consumption, which is a function of the path length, average slope and the instantaneous loading conditions of the truck. A fuel consumption model for waste collection trucks was given by Tavares et al. (2009).

According to Tavares et al. (2009), the fuel consumption during waste collection and transportation depends on the distance travelled, vehicle load and road gradient as well as the actual operating conditions. Tavares et al. (2009) suggest the use of the method proposed by Ntziachristos and Samaras (2000) for the evaluation of these effects to fuel consumption. In this method, the basic fuel consumption per kilometer as a function of speed is expressed by the following empirical formula:

$$FCS = 1068.4V^{-0.4905}, \tag{3.15}$$

where V is the velocity of the truck in km/h.

The empirical formula for the basic fuel consumption is calibrated for half-loaded trucks. The effect of load and road gradient is introduced to the equation by means of two dimensionless correction factors: Load Correction Factor (LCF) and Gradient Correction Factor (GrCF).

$$LCF = 1 + 0.36\frac{(LP - 50)}{100} \tag{3.16}$$

$$GrCF = 0.41e^{0.18x} \tag{3.17}$$

Where LP is the load percentage of the truck and x is the slope percentage. The overall fuel consumption is estimated by means of the following formula, in terms of galloons.

$$fc = FCS \times LCF \times GrCF \tag{3.18}$$

In this study, the road gradient (slope) and vehicle load is introduced to the model by means of a penalty coefficient for the distance travelled. For the calculation of the load percentage of the truck, the average amount of garbage per service location is assumed to be 50kg and the weight capacity of the vehicle is assumed to be 8tons. Therefore, each service point visited is assumed to fill the truck with a percentage of 0.625%. The Load Correction Factor (LCF) can therefore be expressed as:

$$LCF = 1 + 0.36\frac{(0.625n - 50)}{100} \quad (3.19)$$

where n denotes the number of garbage collection service points visited so far by the particular truck. The empirical equation for the gradient correction factor is valid for slopes within the range of -13.5° to 13.5° (-15% to 15%). Converting the road gradient from percentage to degrees, the penalty coefficient can therefore be written as:

$$p(n,\lambda) = 0.41\left[1 + 0.36\frac{(0.625n - 50)}{100}\right]e^{0.2\lambda}, -13.5 < \lambda < 13.5 \quad (3.20)$$

Figure 3.5 illustrates different penalty coefficients for road gradient and loading conditions. The chart demonstrates the importance of slope and load for fuel consumption. For instance, the loading percentages after visiting 50 and 100 waste collection points are 31.25% and 62.5%, respectively. If the road inclination is 10° and the road length is 200m, the excess load for visiting 100 demand points would result in an excess distance of 68m compared to passing through the same road with half the load.

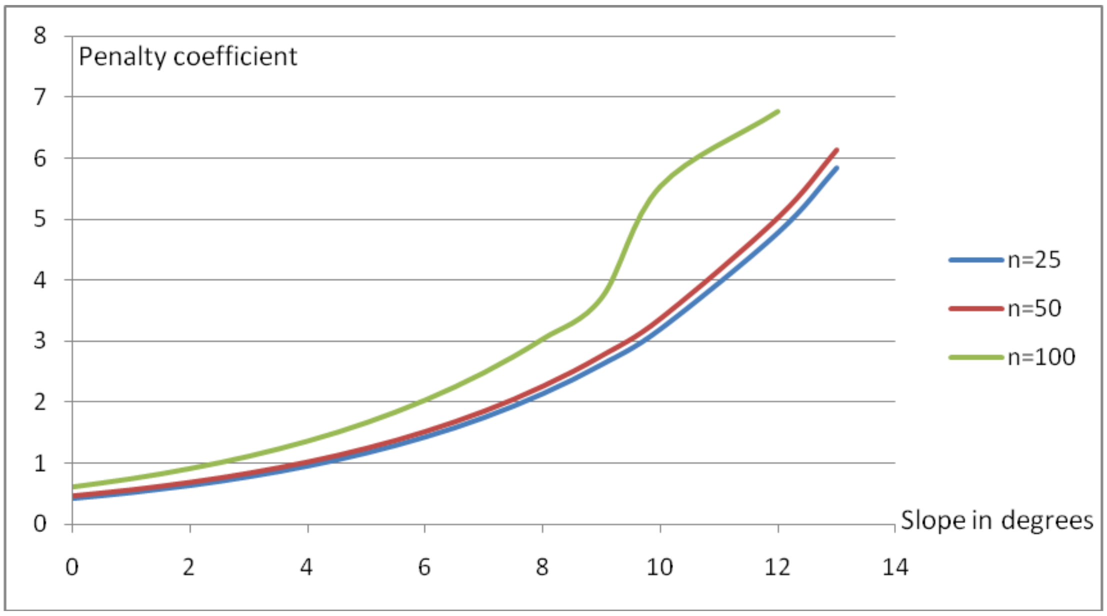


Figure 3.5 – Penalty coefficients for different road gradients and number of pre-visited service points.

Integration of the fuel consumption into the model requires the definition of average slope of edges. λ_{ij}^{1}, which is the average slope of the primary shortest path connecting vertices u_i and u_j. Likewise, λ_{ij}^{2} is the average slope of the secondary shortest path. For simplicity in the notation, the average slope can also be written as $\lambda(path\ no, i, j)$ where path no indicates whether the first or second shortest paths are used, and i and j denote the source and destination vertices, respectively. The definition of the fuel-consumption based cost for the second model becomes:

$$C_{FC}{}_{i}^{k} = D_i^k \times p(i, \lambda(y_i^k, u_i, u_j)). \tag{3.21}$$

The total cost of each route k can be defined as

$$c_{FC}(R_k) = \sum_{i=1}^{n_k} C_{FC}{}_{i}^{k} \tag{3.22}$$

The objective of the model is to

$$Minimize \sum_{k=1}^{m} c_{FC}(R_k) \tag{3.23}$$

Subject to

$$R_i \cap R_j \in \{u_0\}, \quad for\ \forall i,j \in M, i \neq j \tag{3.24}$$

$$\sum_{k=1}^{m} n_k = n \tag{3.25}$$

$$\sum_{i\in N} x_{i0} = \sum_{j\in N} x_{0j} = m \tag{3.26}$$

$$\sum_{i\in N} x_{ir} = \sum_{j\in N} x_{rj} = 1, for\ \forall r \in N \tag{3.27}$$

$$x_{ij} \in \{0,1\}, \forall i,j \in N \tag{3.28}$$

$$y_i^k \in \{1,2\}, \forall k \in M, i \in \{1,2,\ldots,n_k\} \tag{3.29}$$

where

R_k is the route vector for vehicle k, consisting of vertices,

n_k is the number of vertices in route k,

x_{ij} indicates whether there is an edge between vertices i and j,

y_i^k indicates whether the first or second shortest path is used in the i^{th} edge of k^{th} route.

The objective function, Equation 3.23 is to minimize the total distances travelled. Equation 3.24 restricts the routes so that no two vehicles can service the same waste collection location. Equation 3.25 ensures that the sum of total number of waste collection points assigned to each vehicle adds up to the number of waste collection points. Equation 3.26 says that the landfill is entered and left as many times as the number of vehicles. Equation 3.27 shows that each waste collection point is entered and left only once. Equation 3.28 says that there either exist or not exist an edge between points i and j by restricting the possible values of the variable to 0 and 1. Equation 3.29 restricts the paths that can be used to primary and secondary shortest paths.

3.4.2 Optimization Method

In Chapter 2, three broad classes of optimization algorithms were presented: exact, heuristic and metaheuristic. Even though exact methods can the problem to full optimality (i.e., they can find the best possible solution), they can only solve problem instances with up to 100 service points (Fisher et al., 1997). Heuristic methods operate based on inspection and their search capability is limited. Therefore, it is more suitable to consider metaheuristic algorithms as they make a deeper exploration of the search space.

Genetic Algorithms (GA) are selected among other optimization methods. Genetic Algorithms are in the class of Metaheuristic Optimization Algorithms and they are modelled based on the evolution of species. According to Sengoku and Yoshihara (1998), Genetic Algorithm "does not require detailed knowledge of the problem, it can search globally and it can adapt the changing conditions of the problem". Furthermore, it is easy to make the genetic algorithm cooperate with other heuristics to provide a hybrid solution. A detailed description of the implemented hybrid genetic algorithm is presented in Chapter 4.

3.5 Output

The optimization algorithm generates a routes starting and ending at the landfill for each vehicle. The route is represented by an array of numbers representing the visit order of the waste collection points. The numbers are accompanied by a flag indicating whether the first or the second shortest path is used to connect the points.

The output is displayed on a 3D terrain view using Open Graphics Library (OpenGL, 2010). OpenGL is a general purpose, platform independent graphics library that has a variety of functions for drawing graphics in 2D and 3D. Moreover, OpenGL functionalities are embedded in many graphic cards so that very little CPU power is required to process the screen output. Thereby, the user can navigate through map without while the optimization is in progress without affecting the performance significantly. The application has the capability to visualise the road network, garbage collection points and the route along with a 3D terrain view in OpenGL.

3.6 GIS Integration

The developed software is stand-alone and does not work inside a GIS program and therefore, is loosely coupled with GIS environment. It supports ESRI shape files and a Surfer Grid Files as input. The road network and the garbage collection locations are introduced to the system as shape files. For simplicity in metric distance calculations, the shape files are projected using UTM Projection (Zone 36N) with WGS84 datum. The DEM is optional and introduced to the system as a Surfer Grid File.

Apart from operating on GIS data, the proposed optimization model has a spatial constraint that prevents U-Turns. The constraint requires a geometric vector variable which indicates the starting and ending directions of paths between every pair of service points. The proposed model uses this information to select either the primary or secondary shortest path to travel between any two waste collection points.

3.7 Implementation

The developed software is implemented in C++ programming language using Microsoft Visual Studio 2005 as a stand-alone application. The software is structured based on the Document/View architecture; however the related MFC libraries are only used for display and disk I/O features. It is designed as multi-thread application with the genetic algorithm running in a separate thread. This feature enables the user to pan through the map while the genetic algorithm runs in the background without any interruption.

The genetic algorithm is created almost entirely by using ANSI C++ and the standard template library (STL) for required data structures such as dynamic arrays and hash maps. Therefore, it is possible to extract the genetic algorithm code from the application in order to use in other C++ environments with a few modifications.

The sample-runs of the developed software are made on a 2.20GHz double core Intel processor with 2GB of memory. All other processor-demanding applications are closed before the tests in order to provide the same conditions for every single execution.

CHAPTER 4

DEVELOPED HYBRID GENETIC ALGORITHM

In this study, a hybrid genetic algorithm is utilized for the optimization of the waste collection routes. Genetic algorithm is preferred in this study due to the following reasons:

- GAs can provide feasible solutions at any instance of generation,
- A detailed knowledge of the problem is not required (Sengoku and Yoshihara , 1998),
- It is easy to adapt to changing conditions of the problem easily (Sengoku and Yoshihara , 1998),
- GAs can search globally without getting stuck in local optima (Sengoku and Yoshihara , 1998),
- It is possible to co-operate with other techniques (hybridization),
- GAs can be parallelized with very high efficiency.

In Chapter 2, a literature review of Genetic Algorithms for the vehicle routing problems was presented. The idea behind the Genetic Algorithms is explained and various concepts related with Genetic Algorithms were discussed considering the path representation methods and the use of appropriate genetic operators.

This chapter focuses on the details of the hybrid genetic algorithm developed in this study. First, a general flowchart of the genetic algorithm is presented by explaining the components. Then, the stages of the algorithm including the creation of the initial population, the genetic operators, the hill-climbing techniques and the GA parameters are discussed. The strategies used for parallelizing the genetic algorithm are explained by presenting a chart for speed-up. The genetic algorithm is validated on a set of benchmark problems provided by TSPLIB (TSPLIB, 2010).

A flowchart of the genetic algorithm used is presented in Figure 4.1. The proposed genetic algorithm starts by creating an initial population of candidate solutions (referred to as chromosomes, individuals or members). After the creation of the initial population, the fitness of each chromosome in the population is calculated. As multiple instances of the algorithm run in parallel, the chromosomes migrated from other populations are transferred from the exchange pool to the population. The chromosomes are then sorted based on fitness and the selection operator chooses the chromosomes that are to undergo genetic operations. Then, chromosomes to be transferred to exchange pool are selected based on their fitness using the roulette wheel selection method and copied to the pool to be used by other parallel instances of the GA. The genetic operators are applied to the selected chromosomes of the population and the termination criteria are checked. The iteration continues until the termination criteria are satisfied. The source code of various parts of the genetic algorithm is presented in Appendix B.

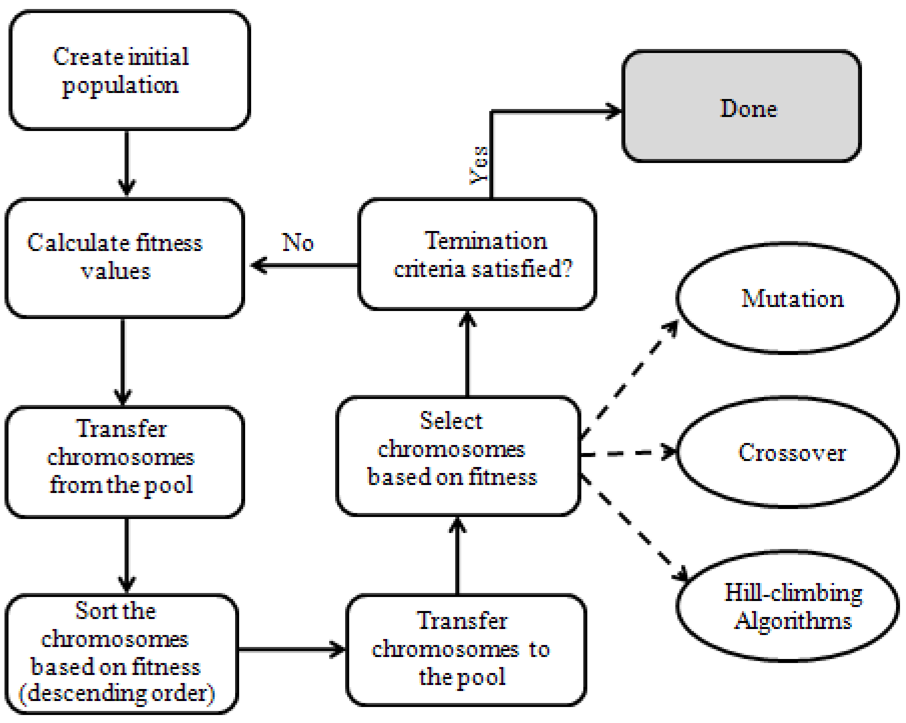

Create initial population
Done
Yes
Calculate fitness values
No
Temination criteria satisfied?
Mutation
Transfer chromosomes from the pool
Select chromosomes based on fitness
Crossover
Sort the chromosomes based on fitness (descending order)
Transfer chromosomes to the pool
Hill-climbing Algorithms

Figure 4.1- Flowchart of the proposed Hybrid Genetic Algorithm

4.1 Essential Elements of GA

The following items are the essential elements of genetic algorithms that must be considered for any genetic algorithm.

- Representation of Chromosomes
- Generation Initial Population
- Fitness Function
- Elitism
- Mutation
- Crossover
- Termination Criterion
- Tuning of Parameters

In this genetic algorithm, path representation is preferred among other representation methods due to its simplicity and applicability to many genetic operators and hill-climbing techniques. In the path representation, each waste collection point is represented by a specific positive number. The route assigned to each vehicle is represented by a vector of numbers.

The chromosomes comprising the initial population are generated independently. The number of chromosomes in a population is referred to as the population size. The procedure below explains how each individual chromosome is generated. Repeating this procedure as many times as the population size, the initial population is generated.

The flowchart for generating an initial chromosome is illustrated in Figure 4.2. Initially, the path vector of each vehicle route in a chromosome is empty and the bucket of available points is full. Hence, the initial transportation costs for each vehicle are zero.

Random waste collection points are assigned to each vehicle as their first destinations and the transportation costs are updated accordingly. Any point assigned to a vehicle is marked as assigned in order to assure that it will not be re-assigned later. Thereby, generation of illegal solutions is prevented. Then, the vehicle with lowest transportation cost is selected in order to assign the next destination. Roulette wheel selection is used to select the next point. The procedure continues until all points are assigned to a vehicle. Since the algorithm uses roulette wheel selection, the initial route construction process is stochastic.

The probability of selecting any available point j is proportional to a constant power of the inverse distance between points. Thus, the probability of being selected for each candidate point j is given in Equation (4.1.

$$\rho_j = \frac{d_{ij}^{-\propto}}{\sum_{k=1}^{N} d_{ik}^{-\propto}} \tag{4.1}$$

where

d_{ij} denotes the distance from the last selected point to the next candidate point and

$\propto$ is the intensification constant.

The higher the intensification constant ($\propto$), the higher the tendency of selecting closer points will be. As $\propto$ approaches to 0, the roulette wheel selection method behaves more like random selection. The value of $\propto$ is kept constant throughout the generation of each individual chromosome; however, its value is increased gradually after each chromosome. The value of $\propto$ starts from 1.0 and reaches to 4.0 in the last chromosome of the population. The variation of $\propto$ values throughout the generation process is to create initial chromosomes of different characteristics.

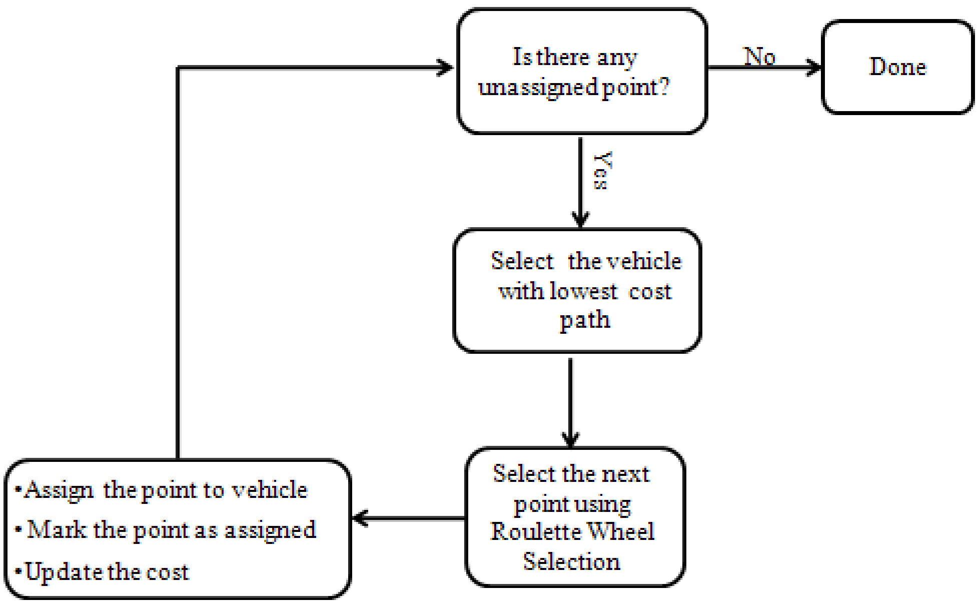

Is there any unassigned point?
No
Done
Yes
Select the vehicle with lowest cost path
Select the next point using Roulette Wheel Selection
•Assign the point to vehicle
•Mark the point as assigned
•Update the cost

Figure 4.2- Creation of an initial chromosome

The fitness function in the Genetic Algorithm is a measure of how the objective function in the mathematical model is satisfied. Therefore, for both optimization models presented in Chapter 3, only the fitness function is modified as the constraints of the problem are the same for both cases.

The fitness of a chromosome is calculated as a function of transportation distance and the penalty component of the chromosome. The penalty coefficient (p_i) is a multiplier of the transportation distance (d_i). The value of the penalty coefficient constant (1.0) for the first optimization model which minimizes the distance travelled. On the contrary, in the second model which minimizes the fuel consumption, p_i represents the penalty coefficient given in the mathematical model. The fitness of each individual is calculated using the formula in Equation (4.2).

$$f_i = (p_i \times d_i)^{-1} \tag{4.2}$$

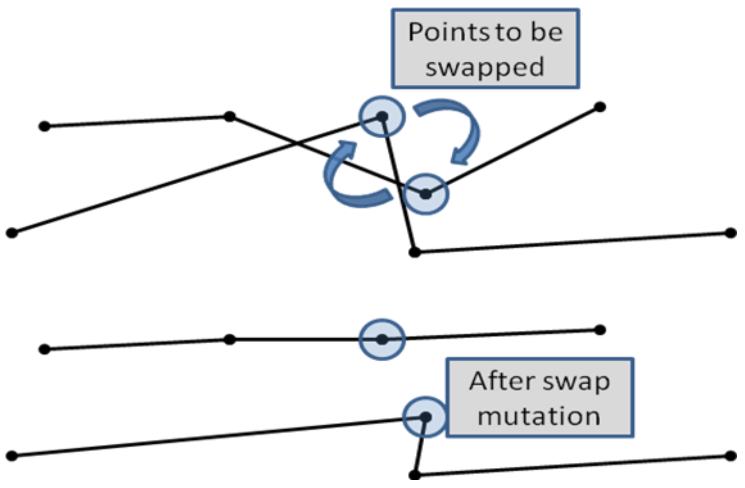
The first 10% of the chromosomes possessing the highest fitness values are protected from deterioration and are said to be in the elitism region. The approach to elitism is different from other studies in that the chromosomes in the elitism region are only protected against modifications which lower their fitness values. In traditional approaches, the elite chromosomes cannot be modified in any manner.

Three different mutation operators are employed in the proposed genetic algorithm: point swap mutation, modified 2-opt heuristic mutation, and insertion mutation. The chromosomes to undergo mutation are selected randomly. For all mutation types, any mutation that is to reduce the quality of the chromosome cannot be applied to the chromosomes in the elitism region. They can only be applied for the chromosomes that are beyond the elitism region in order to increase genetic diversity in the search. The point swap mutation is a simplified version of swap sequence mutation proposed by Nazif and Lee (2010). Two waste collection points are swapped rather than two sequences of routes. However, the points to be swapped are not selected randomly. First, an arbitrary waste collection point in the chromosome is selected. The other point to be swapped with the first one is selected by means of roulette wheel selection such that closer points have a higher probability of being selected for exchange. Then the two points are swapped. Therefore, it is likely that the selected points are close to each other as the selection probability is related with the inverse distance. This mutation supports swapping points between vehicles and is very simple and powerful.

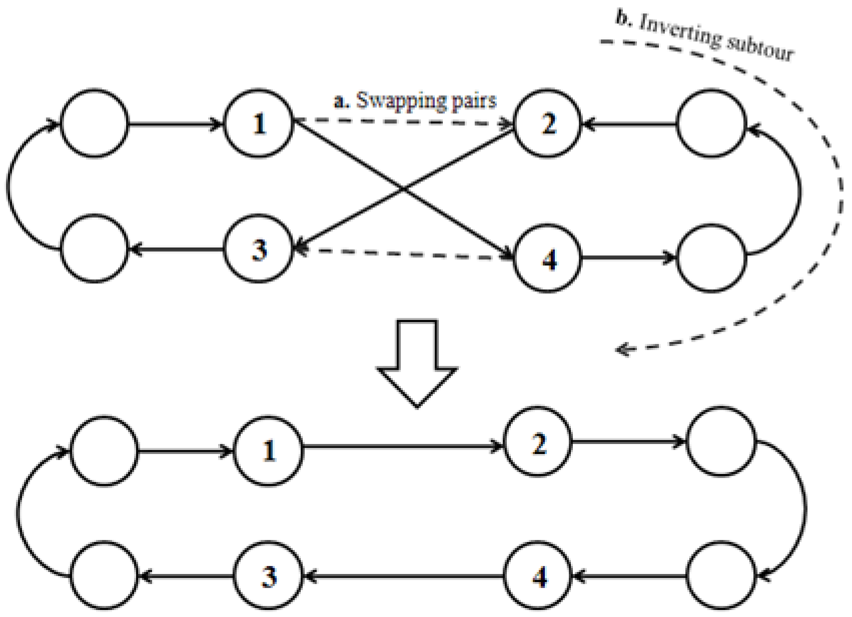
The 2-opt mutation utilized in this study differs from the original one explained in Chapter 2 in that not all the pairs of edges are checked for swapping pairs. Instead, only edges that are closer than the average edge size of the chromosome are considered for 2-opt mutation. The average edge size of the chromosome is calculated by dividing the total path length of the chromosome by the total number of waste collection points. The sequence of points between the swapped pairs is inverted to check if inversion enhances the result.

In the insertion mutation, all edges are checked for a point which is closer to the edge than quarter the size of the edge. If there is such a point within the threshold distance, the point is removed from its original position and inserted between the two points comprising the edge.

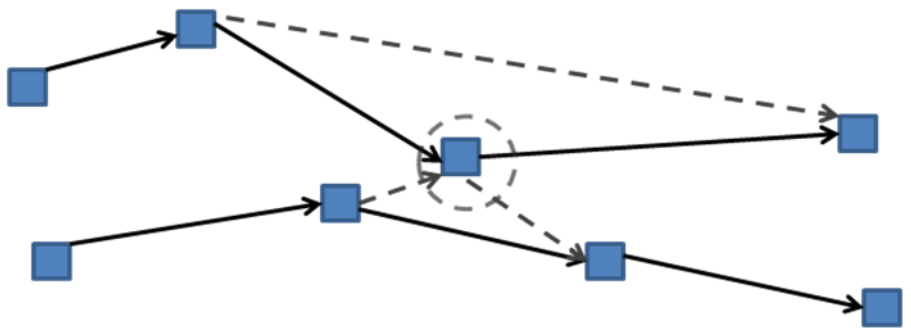
After each mutation, each route is repaired so that second shortest paths are used wherever it is necessary. Figure 4.3 illustrates the mutation algorithms used within this genetic algorithm.



a.) Point Swap Mutation



b.) 2-opt Mutation



c.) Insertion Mutation

Figure 4.3-Mutation Algorithms

Edge Recombination Crossover (ERX) is used to generate new offsprings from parent chromosomes. Although alternative crossover algorithms such as PMX and OX could also be used, they are not as effective as ERX in terms of inheriting important genetic material such as edges. As ERX (Whitley et al., 1989) was originally proposed for the Travelling Salesman Problem (TSP), it is slightly modified to support multiple vehicles.

To start with, the edge list (Table 2.3) is constructed for all points by using two parent chromosomes as explained in Chapter 2. Then, all vehicles are assigned an initial point from either one of the two parents. At each iteration, the vehicle with lowest transportation cost is selected and the next destination of the vehicle is selected from the edge list. If the edge list contains more than one edge starting from the current point, the shortest edge (nearest point) is selected. The process continues until all points are used.

The termination criterion for the genetic algorithm in this study is based on the number of generations without further evolution of the best cost obtained. To determine a suitable number of inefficient generations required to terminate the optimization process, initially the termination criterion is adjusted so that the genetic algorithm terminates after 5000 inefficient generations. After running the algorithm with 20 runs with well-tuned parameters, the number of generations passed between the last two enhancements is recorded. For instance, if a genetic algorithm makes the last cost enhancement in the generation 15000 and the previous enhancement in the generation 14000, the difference 1000 is recorded as the result of that particular run.

Figure 4.4 illustrates the frequency histogram for the number of generations between last two enhancements for different runs with a bandwidth of 100. Among the 20 samples, only a single sample continued evolving after 1746 generations. It can be inferred from the chart that almost all of the sample runs cannot demonstrate any further enhancement after 1000 generations without evolution. The proposed genetic algorithm terminates if there is no further enhancement of the best route in the last 2000 evolutions.

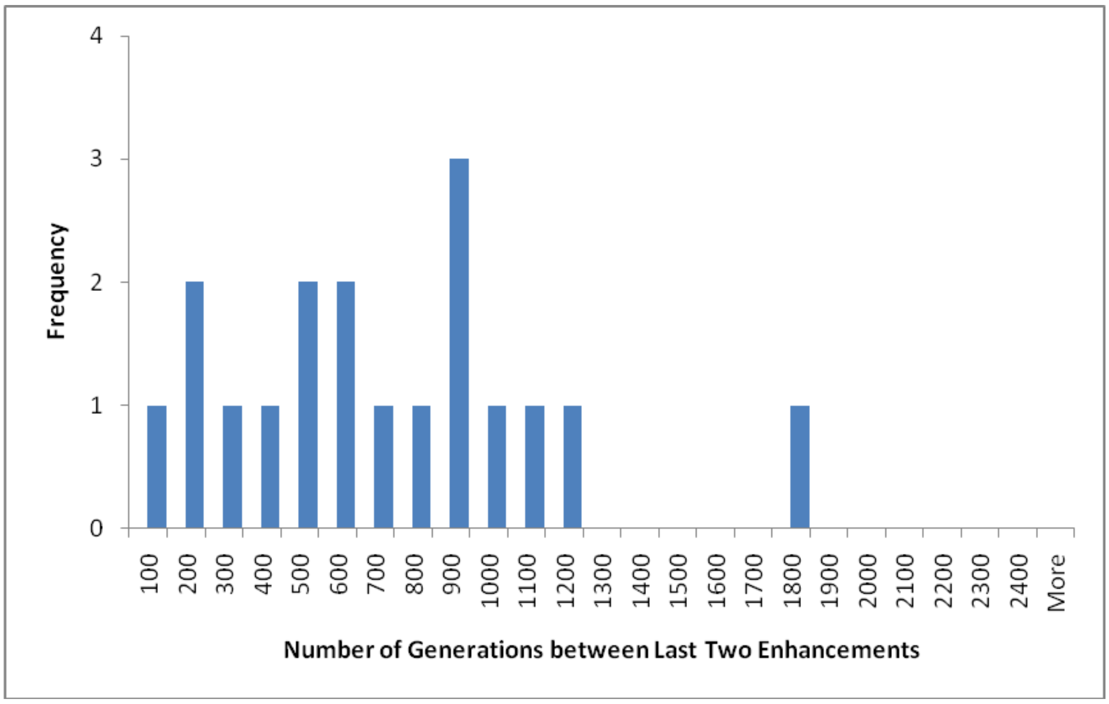


Figure 4.4 – The frequency histogram for the number of generations between the last two enhancements for 20 test runs.

Metaheuristic optimization methods are widely used in optimization problems with large inputs and almost always require a fine tuning of parameters. Tuning the parameters is of great importance in genetic algorithms as the evolutionary process converges to better results in a shorter amount of time with the related parameters well-adjusted. The main parameters involved in genetic algorithms are the percentages of individuals that are to undergo mutation and crossover as well as the population size.

Figure 4.5 illustrates the best route distances obtained by running the genetic algorithm with different mutation and crossover percentage values. The genetic algorithm was run with 3 different mutation percentage combinations ranging from 5% to 15% and 9 different crossover percentage combinations ranging from 40% to 80%. The experiment was done three times and average costs are used in order to reduce the chances of obtaining misleading coincident results. According to the experiment using 27 different mutation-crossover combinations, the best results were obtained by using a crossover percentage of 45% and a mutation percentage of 5%.

However, this combination makes a local peak in the chart with a small variation of parameters resulting in a bad performance, mutation values of 15% with 50% to 60% of crossover is also recommendable.

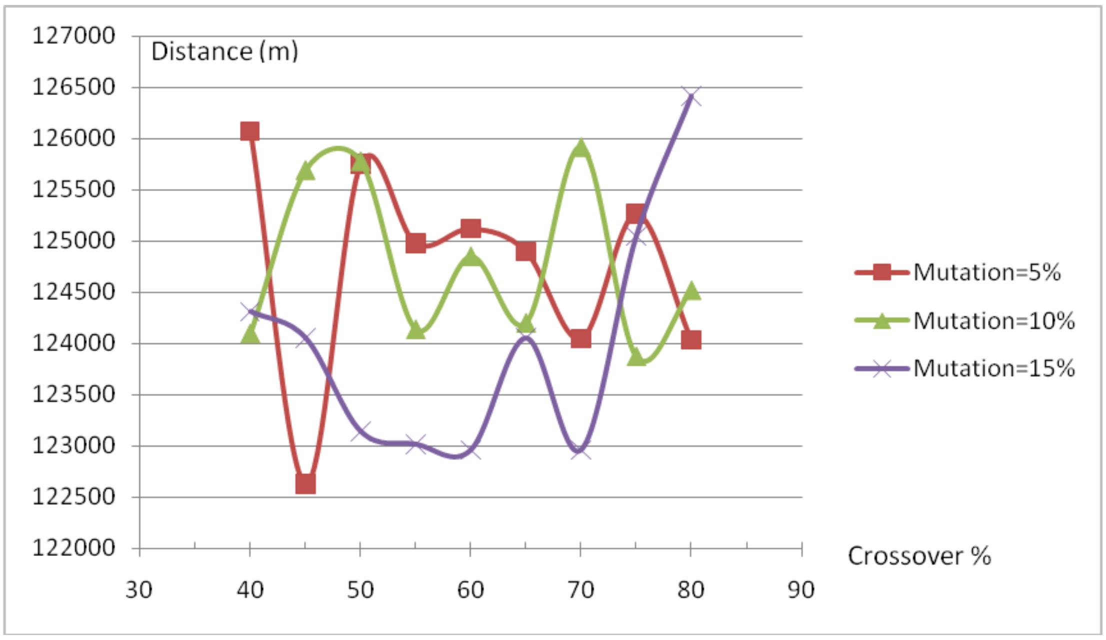


Figure 4.5 – Best transportation distances obtained by trying different crossover and mutation combinations (without hill-climbing algorithms).

Selection of the most suitable population size is also of great importance. If the population size becomes too low, the genetic diversity would not suffice to sustain the evolution and the evolution of potential solutions would seize quickly. Likewise, if the population size becomes too high, it would take longer to execute the genetic operators as well as the hill-climbing algorithms.

Figure 4.6 illustrates a range of different population sizes used with the genetic algorithm. The genetic algorithm is run on 20 different population sizes, ranging from 150 to 1050. The experiment was repeated 3 times and the average distance values are used. The best results were obtained by using a population size of 500.

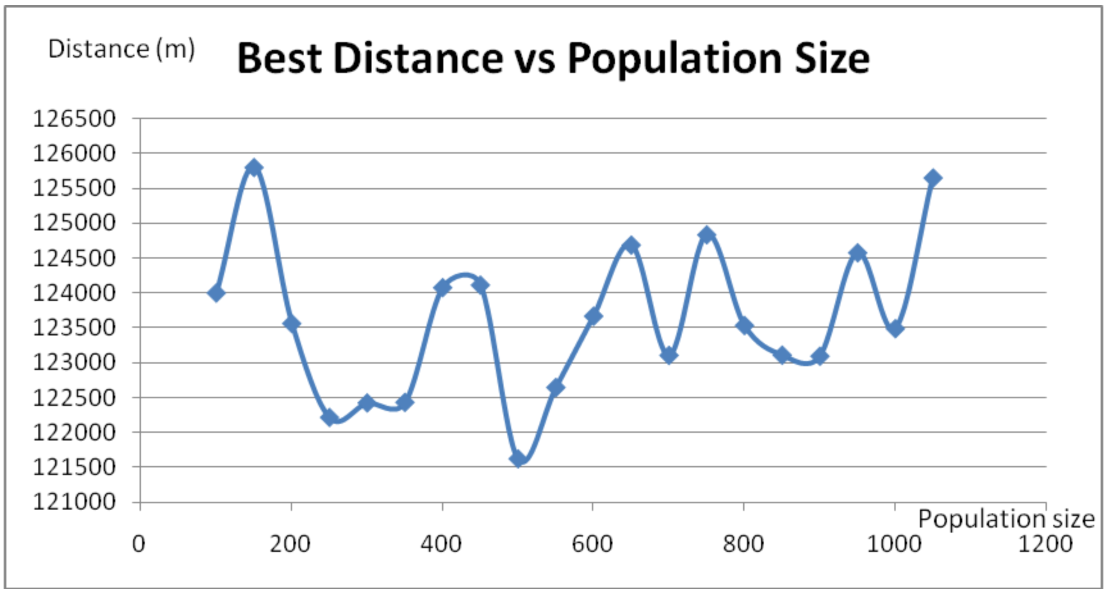


Figure 4.6 – Best transportation distances obtained by selecting different population sizes

4.2 Hill-climbing Algorithms

Hill-climbing is a local search method that is often used in cooperation with genetic algorithms to increase the performance. Hill-climbing algorithms are iterative and their aim is to gradually improve the solution by enhancing a small part of it at each iteration. The genetic algorithm developed in this study is called “hybrid” as it utilizes hill-climbing algorithms that aim to enhance the evolutionary process.

The 2-opt mutation described in the preceding sections is a hill-climbing algorithm as it attempts to find better solutions each time by switching pairs between different sets of edges. If the change results in a better solution, an incremental change is made and the process continues until no further enhancement is possible.

Another hill-climbing method used in this study is exhaustive search. It is practically impossible to find the best permutation of a large set of points with exhaustive search (also referred to as brute-force search) as the computation time grows exponentially with the number of points. However, exhaustive search can be used to find the best possible arrangement of points in the route for a small part of the route. For each route, a small part consisting of 8 to 12 nodes is randomly selected at each generation to apply the exhaustive search. The best permutation of the selected range nodes is found and replaced with the old sequence.

Purely exhaustive search checks for every possible arrangement of points and hence, is inefficient. To speed up the search, alpha beta pruning and heuristic methods are employed. The alpha beta pruning technique stops the evaluation of a permutation even before the sequence is completed when the instantaneous cost of the partial sequence exceeds the lowest cost found so far. When the evaluation of a permutation is stopped, any dependant permutation will not be considered. This approach will narrow down the search space without affecting the overall result. The heuristic improvement method on the other hand focuses on obtaining a feasible (low-cost) permutation as soon as possible so that the alpha beta pruning technique can eliminate the non-optimal permutations earlier. This can be achieved by sorting the points so that the nearest point is checked first in the search algorithm.

4.3 Parallelization

Genetic algorithms are computationally costly considering the efforts to apply the genetic operators and evaluation functions. However, they are very suitable for parallelization. Distributed computation increases the performance of the genetic algorithm significantly.

The genetic algorithm is parallelized by the course-grained PGA islands model (Shengjun et al., 2008). In this model, multiple instances of the genetic algorithm run independently on different processors. Each instance of genetic algorithm is fed by the same initial data and represents an evolving subpopulation. Each subpopulation evolves independently on their processor in parallel executing the same genetic algorithm. Since every individual subpopulation is isolated and the only means of genetic transfer between different subpopulations is migration, each subpopulation is associated with an imaginary island. Figure 4.7 illustrates the implementation of the model in the genetic algorithm.

As the instances of genetic algorithm discover new routes, 5 individuals from every subpopulation are duplicated and sent to the exchange pool every 100 generations. Likewise, every subpopulation receives 3 individuals from the exchange pool every 50 generations. The amount of individuals currently existing in the pool is less than 50, all of them are pulled out of the pool. The individuals in the subpopulation are not allowed to return to their original islands once they are sent to the pool.

The access to the exchange pool is sequential. Therefore, only one island can access the exchange pool at any instance. Whenever an island is to send chromosomes to the exchange pool, first it locks the pool if it is available. Once the chromosome is migrated, it is unlocked. This prevents concurrent access to the pool, which may corrupt the chromosomes.

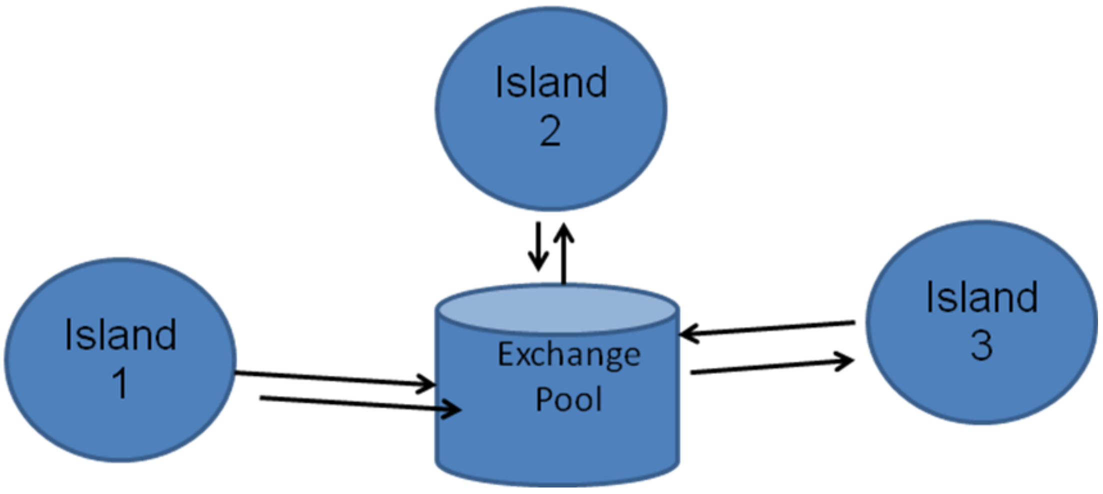


Figure 4.7 – Illustration of the Island model

For the parallelization of the genetic algorithm, the OpenMP library is used. OpenMP (Open multi-processing) is a cross-platform application programming interface that supports multi-platform shared memory multiprocessing programming in many languages on a variety of architectures, including UNIX and Microsoft Windows platforms. It has a number of compiler directives, library routines and environment variables that assist the programmer to parallelize the routines that require high speed.

In parallel programming (Parallel Programming Lecture 14 Computer Performance, 2010), the term ‘speed-up’ refers to the ratio to which a parallel algorithm is faster than a corresponding sequential algorithm. It is calculated by dividing the time passed for accomplishing a task using a sequential algorithm by that using a parallel algorithm (4.3). Efficiency, on the other hand, is a measure of how effectively the processors are used. It is calculated by dividing the speed-up by the number of processors (4.4).

$$S_p = \frac{T_1}{T_p} \tag{4.3}$$

$$E_p = \frac{S_p}{p} \tag{4.4}$$

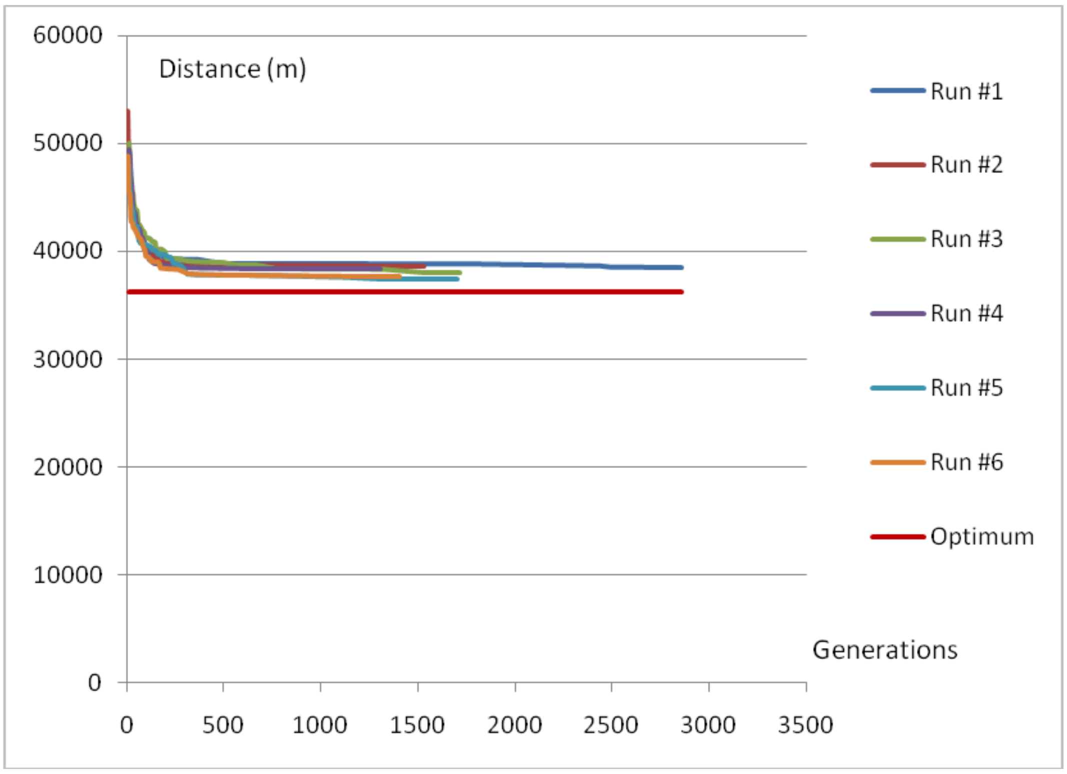
4.4 Validation of the Genetic Algorithm

The validation of the proposed genetic algorithm is made by running the software on three sample asymmetrical TSP benchmark problems provided by TSPLIB (TSPLIB, 2010). Each benchmark problem is given a name indicating the number of service points. The benchmark problems are provided as a text file containing a header part and a distance matrix. The header includes information about the dimensions of the particular problem and the cost of best possible solution (optimum distance) calculated before using exact methods. The distance matrix provides the distances between every pair of nodes. In the validation stage, the program directly uses the distance matrix read from the sample problem file.

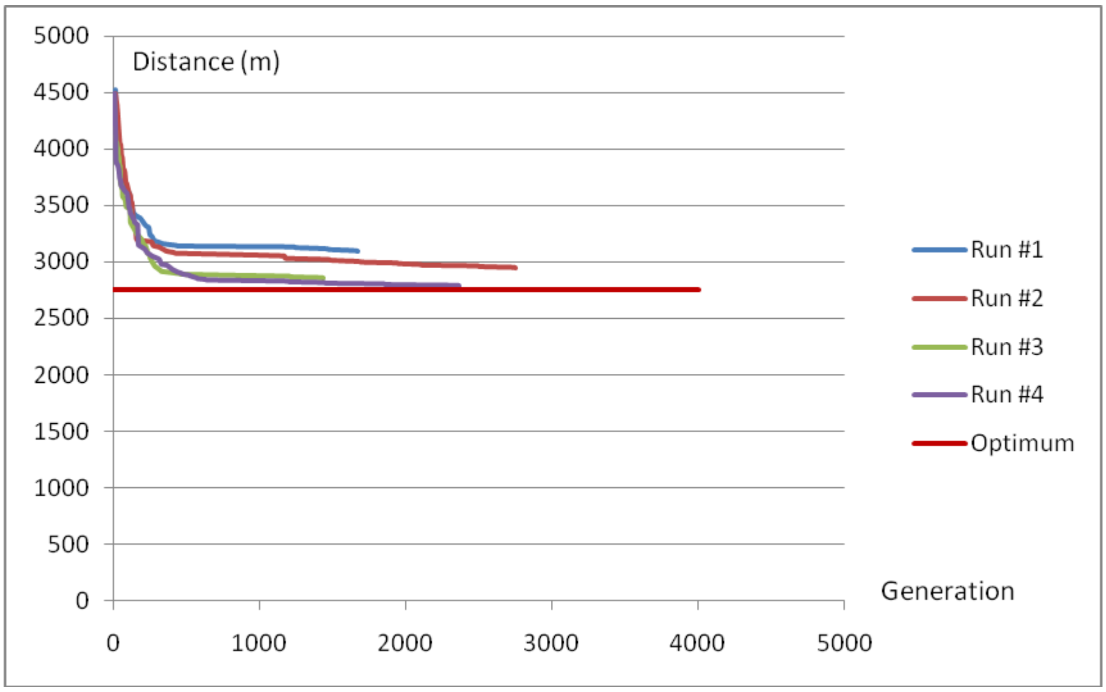
Table 4.1 - TSPLIB Benchmark Results

Problem Name	Number of Nodes	Optimum Cost	Cost Obtained	Deflection from Optimum (%)
kro124p	124	36230	37374	3.15
flv170	170	2755	2784	1.05
rbg358	358	1163	1191	2.40
rbg443	443	2720	2756	1.32

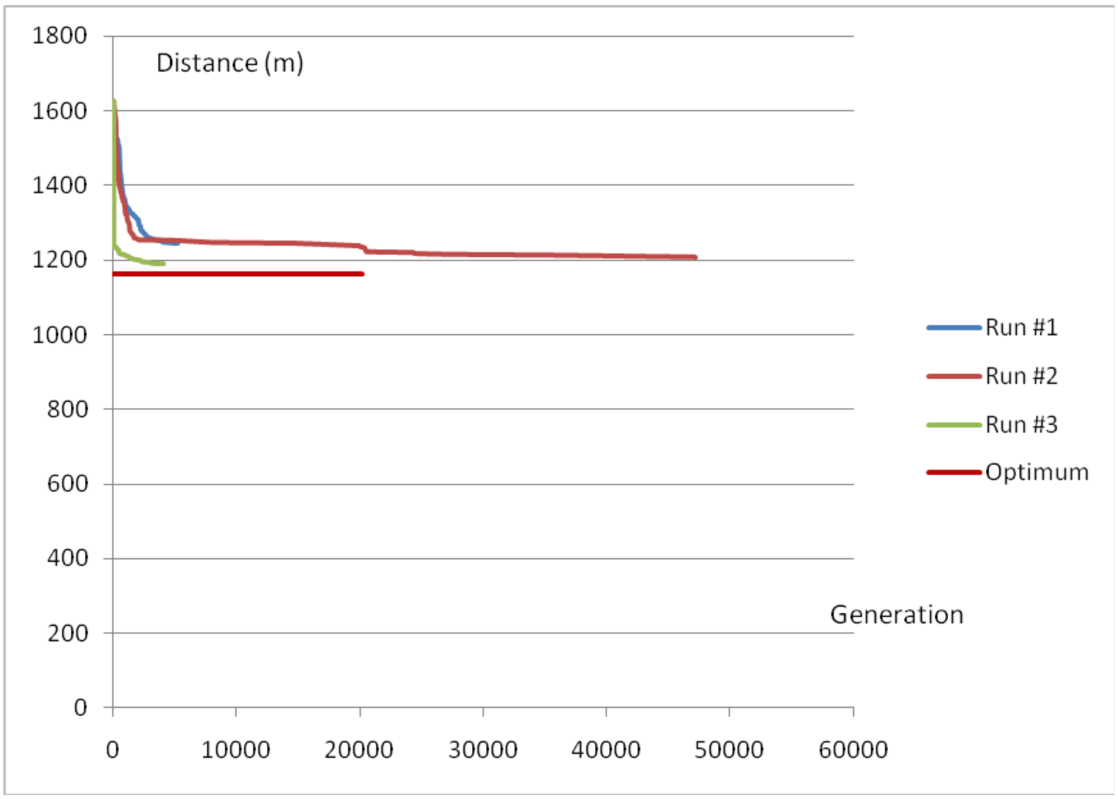
The progress of the solution of the benchmark problems is demonstrated in Figure 4.8. The x axis shows the number of generations of the genetic algorithm to reach to the distance represented by y axis.



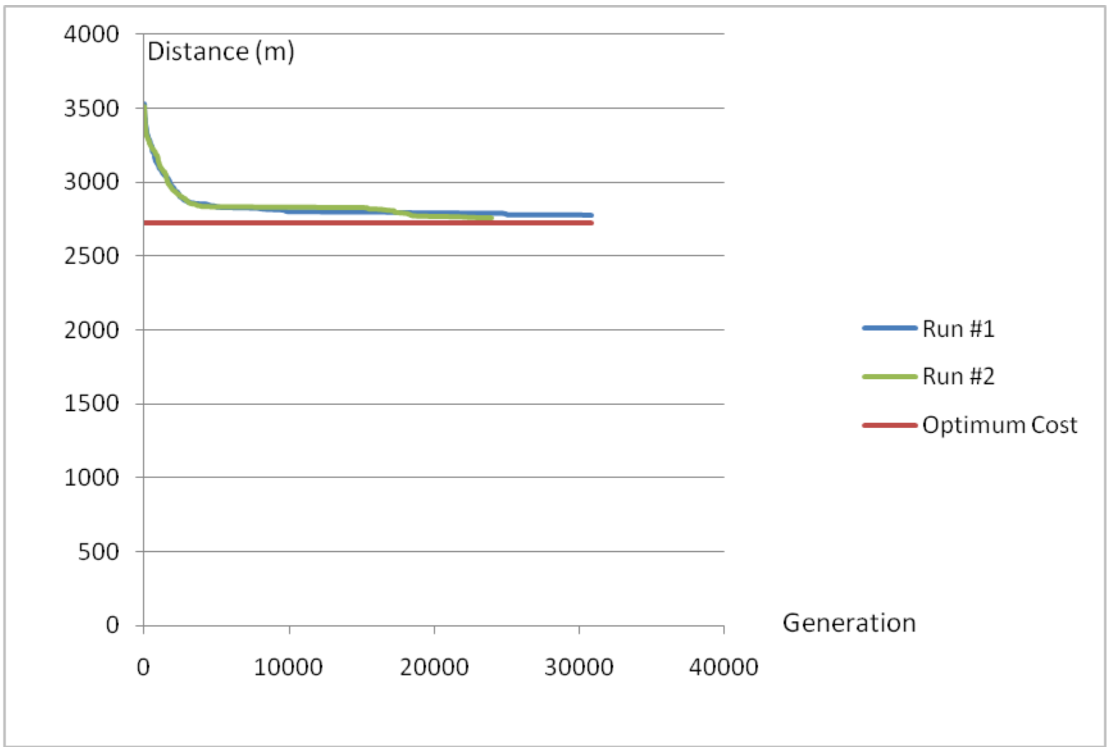
a.) Problem Name: kro124p (3.15% deflection from optimal solution)



b.) Problem Name: flv170 (1.05% deflection from optimal solution)



c.) Problem Name: rbg 358 (2.40% deflection from optimal solution)



d.) Problem Name: rbg 443 (1.32% deflection from optimal solution)

Figure 4.8- Distance vs Generation Graphs for Validation Problems

The benchmarking results show that the proposed Parallel Hybrid Genetic Algorithm solves the sample benchmark problems within 1.05 to 3.15 optimality within less than 5 minutes. Therefore, the algorithm is reliable and is capable of searching globally.

4.5 Parallelization of the Genetic Algorithm

The chart in Figure 4.9 illustrates the efficiency of the parallelization by comparing the number of generations per millisecond in single and double processor case. The test was made in exactly the same conditions by closing all other applications that have a potential to affect the results. Comparing the slopes of the single and double processor cases, the speed-up is calculated as 1.83 with an efficiency of 91.5%.

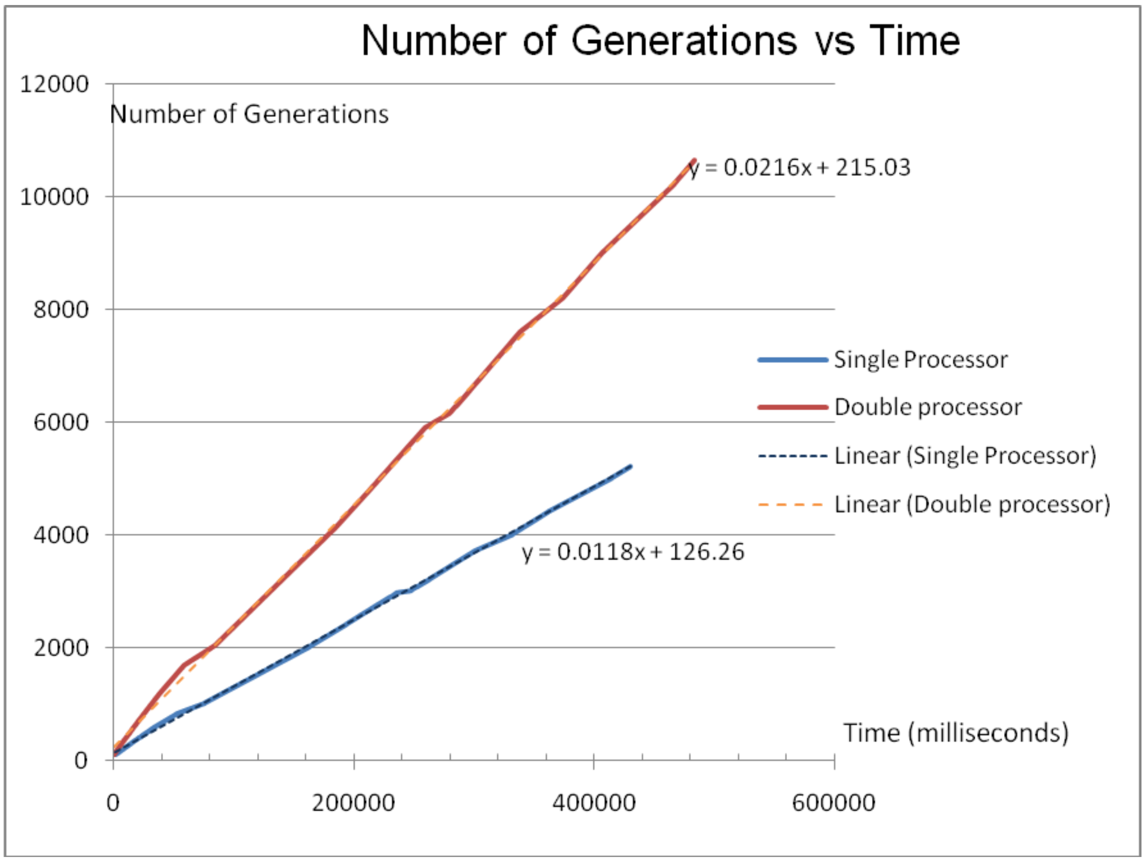


Figure 4.9 - Comparison of single and double processor case. The speed-up obtained by using 2 processors is 1.83 and the efficiency is 91.5%.

CHAPTER 5

CASE STUDY

5.1 Study Area

The proposed area of application for the solid waste collection routing problem includes Bahçelievler and Emek neighbourhoods in Çankaya/Ankara/Turkey with an approximate area of 2.5km^2. The altitude of the region ranges from 860m to 920m above sea level. The area is divided into three zones: Aşağı Bahçelievler, Yukarı Bahçelievler and Emek. For the current waste collection operation; three vehicles are allocated, one for each zone. The waste collection routes are determined based on driver's experience (Uraz, 2002). Following collection waste is disposed at ITC Mamak Landfill Site.

Even though the study primarily focuses on the routes generated in Bahçelievler and Emek Neighbourhoods, the available map data covers entire Ankara in terms of major and main roads and also all the streets of Bahçelievler and Emek Neighbourhoods. Bahçelievler and Emek Neighbourhoods are located approximately at the center of the city whereas the Mamak ITC Landfill is located in the South-Eastern part of the city, bounded by Natoyolu Street from East and the Ankara Ring Road from the South (Latitude: 39.878259, Longitude: 32.934608). The maps are obtained as satellite images in the form of GeoTIFF and then digitized using appropriate GIS tools.

Figure 5.1 illustrates the location of the study area and Mamak Landfill in Ankara. The points demonstrated by asterisks correspond to the waste collection locations. In this study, 239 hypothetical waste collection locations are digitized homogenously. These locations are assumed to be the collection points for the wastes produced in the households situated on the relevant streets.

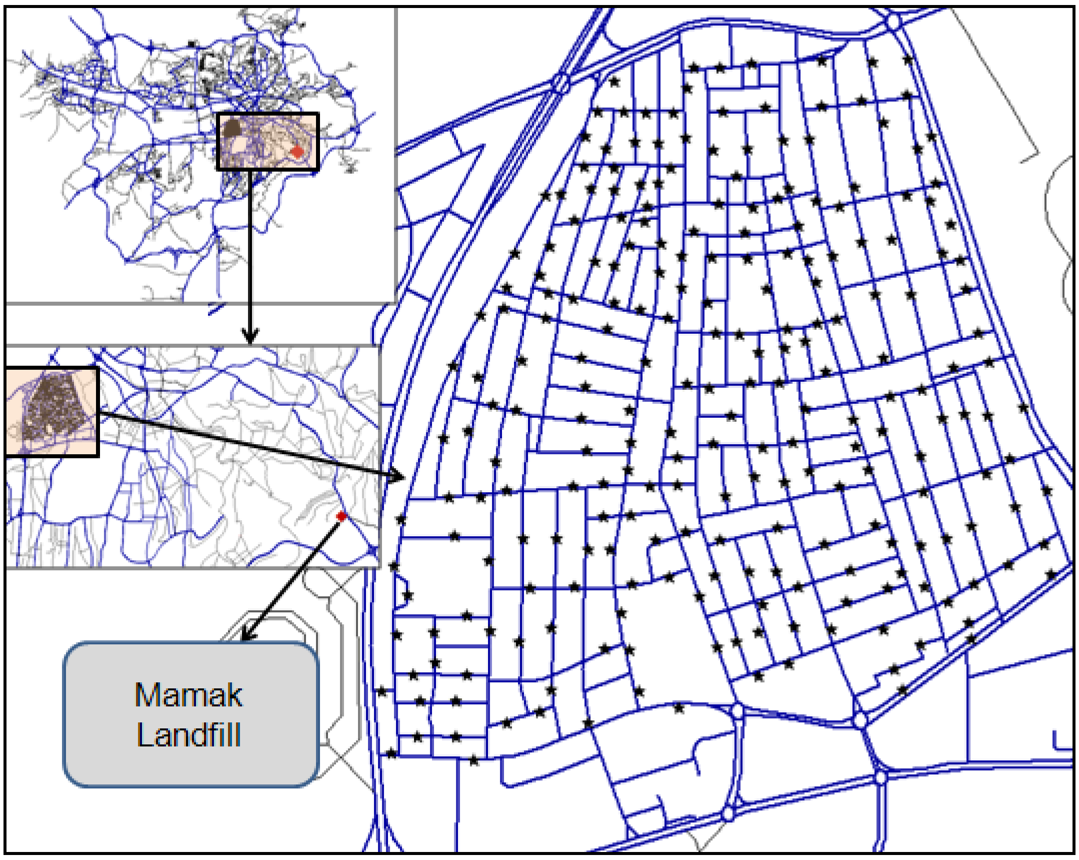


Figure 5.1 – The Study Area.

In Figure 5.2, a terrain view of the Bahçelievler and Emek Neighbourhoods is provided in order to illustrate the varying altitudes. The locations of the waste collection points are marked with red dots. The altitude increases from yellow to dark green. The main trends of slope in the area are towards East and West from the ridge defined by the 4th street (Figure 5.2) and are indicated by arrows.

The road inclination trends are particularly important as far as the optimization model which aims to minimize the fuel consumption is concerned. The fuel consumption model used in this study (Tavares et al., 2009) depends on instantaneous vehicle load and the road inclination.

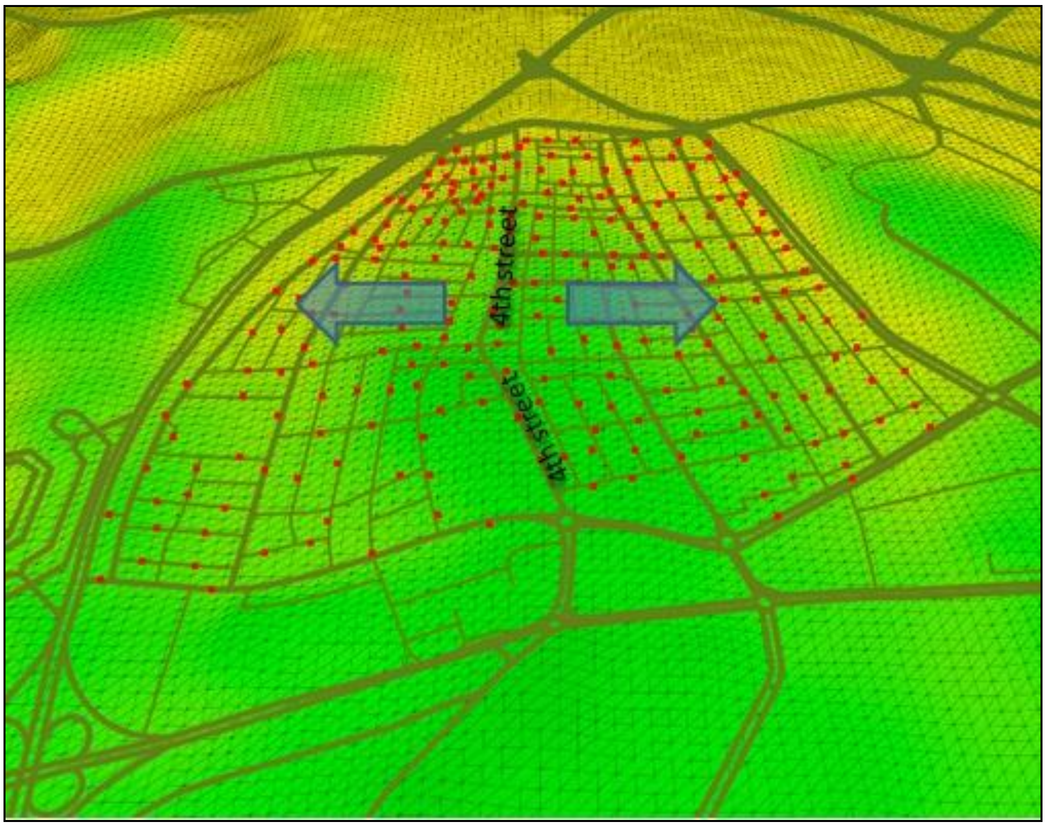


Figure 5.2 – The study area in 3D Terrain View. The slopes are indicated by arrows.

The unidirectional and bidirectional streets in the study area are as illustrated in Figure 5.3. The streets that are shown with red are bidirectional streets. Conversely, other streets shown in blue are unidirectional. The direction of traffic flow is marked by an arrow. Unidirectional roads are important constraints in planning the routes as they have the potential to change the vehicle routes significantly.

The digital elevation model used to construct the 3D terrain is loaded from a Surfer grid file (Surfer, 2010). In order to prepare the grid file, a 90m resolution SRTM grid of Ankara is used along with 70 additional altitude samples of Bahçelievler and Emek Neighbourhoods taken from Google Earth. The final grid of 30m resolution is generated by means of kriging interpolation. The interpolation is made by using Surfer (Surfer, 2010).

The datum and the projection of the shape files are WGS84 and UTM 36N (Universal Transverse Mercator Projection, Zone: 36N), respectively. For convenience in measuring distances, metric system is used. The coordinates of the SRTM altitude data has also been projected using the same parameters before the interpolation.

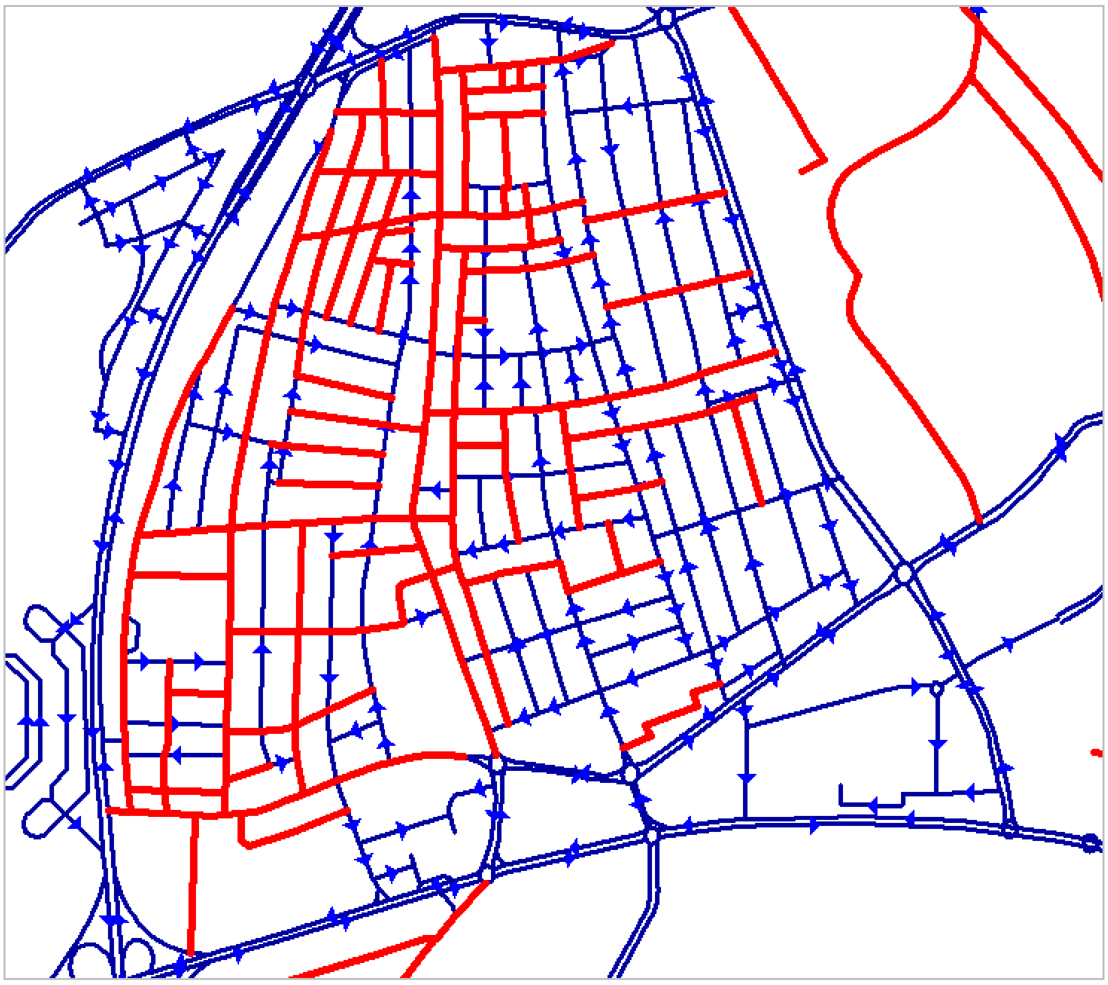


Figure 5.3- Road directions in the area. Roads marked with red indicate bidirectional roads whereas blue roads are unidirectional.

The landfill in Mamak and 239 hypothetical waste collection locations within the study area are represented by a point-featured shape file. The altitude data of the terrain is prepared by interpolating the freely available SRTM (SRTM, 2010) data (90m resolution) with altitude samples collected from Google Earth using Surfer (Golden Software – Surfer, 2010). The resultant grid is saved with grid size of 30m.

5.2 Results and Discussions

Two different optimization models are implemented for the optimization algorithm. The first one aims to minimize the total distance travelled and the second one aims to minimize the total fuel consumption. In the end, is aimed to demonstrate that the two approaches produce different results regarding their objective functions.

In Chapter 4, the genetic optimization algorithm was validated by using 4 TSP Benchmark Problems. Even though this provides some degree of reliability for the algorithm, it is not sufficient. The proposed optimization model is designed to support spatial constraints and therefore, it has to be tested using small test problems in real road network. The first optimization model which minimized the total distance travelled is tested on small test problems within Bahçelievler and Emek Neighbourhoods. The results of the test problems (generated routes) can be verified by inspection easily. Testing the optimization algorithm on the real road network with unidirectional and bidirectional roads, the following items are verified:

- Closure of the path
- Avoidance of U-Turns
- Compliance with road directions
- The shape of the route path

In the first test case, 10 waste collection points are digitized within the study area randomly. The locations of the waste collection points are shown in Figure 5.4.

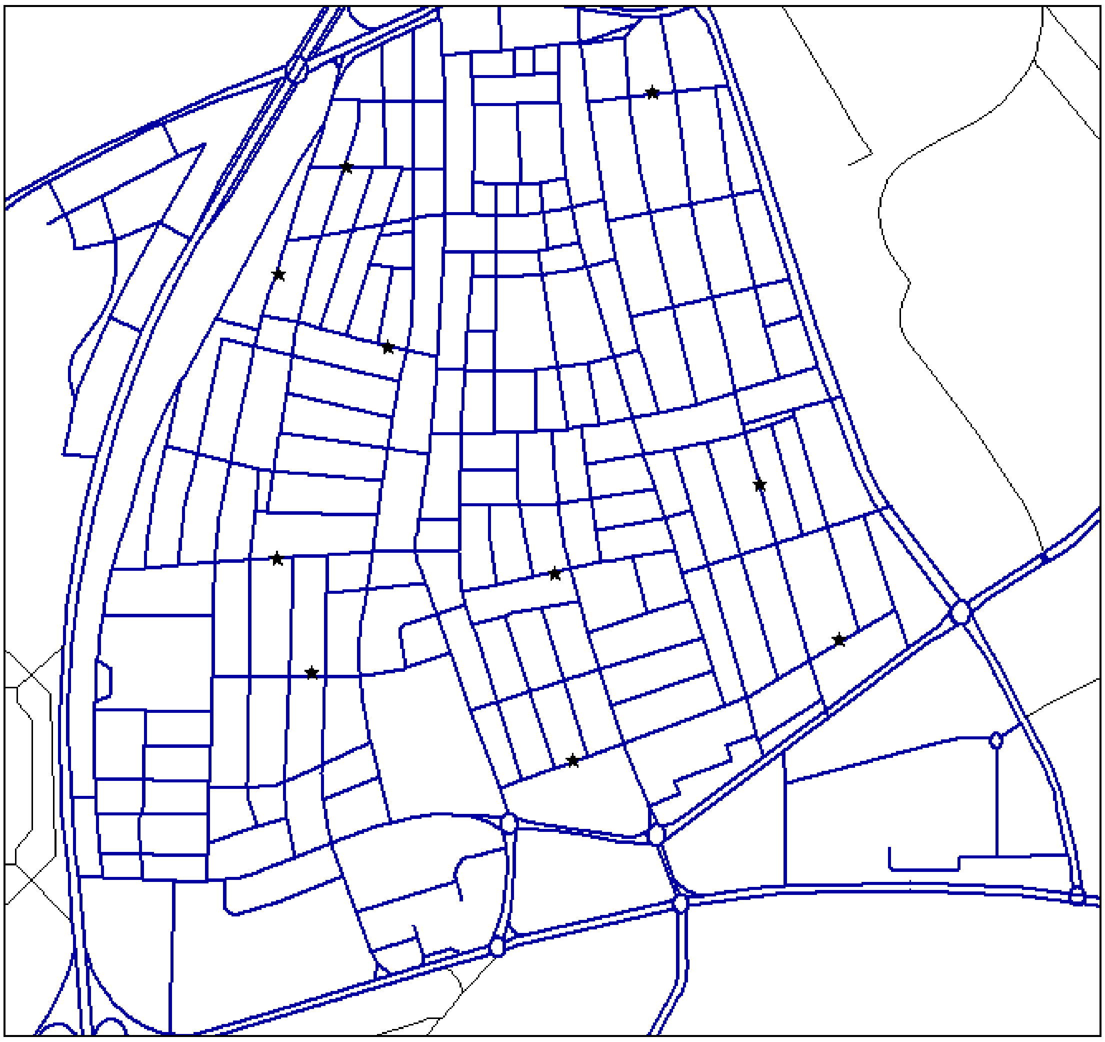


Figure 5.4 – The Distribution of Waste Collection Locations in Test Case 1

For this test case, the routes generated are analyzed for two cases:

- Single Vehicle (Figure 5.5 and Figure 5.6)
- 2 Vehicles (Figure 5.7 and Figure 5.8)

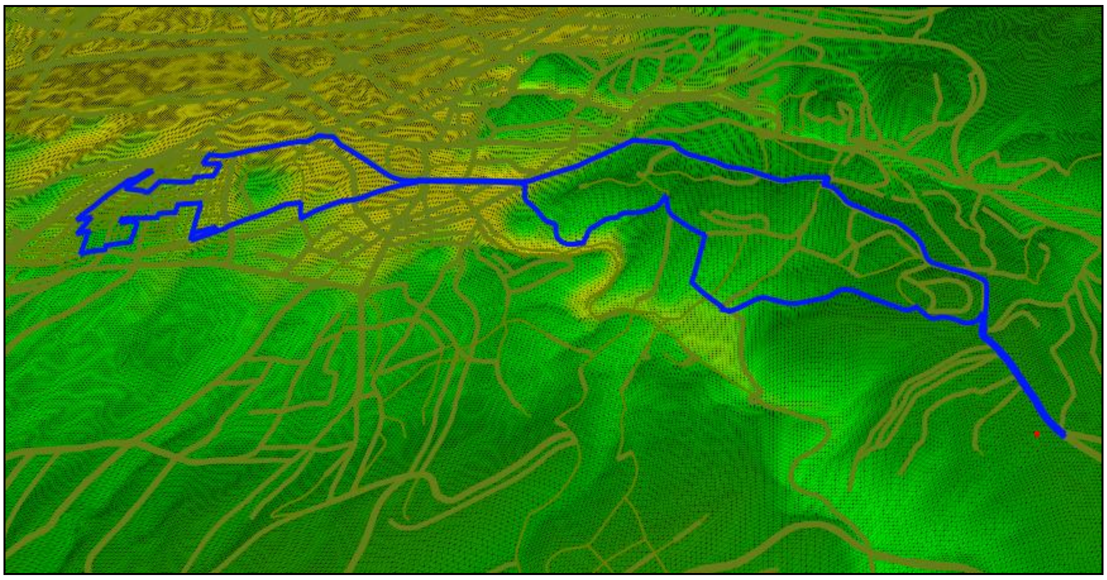


Figure 5.5 - Test Case 1 - A Full View of the Route Generated (Single Vehicle)

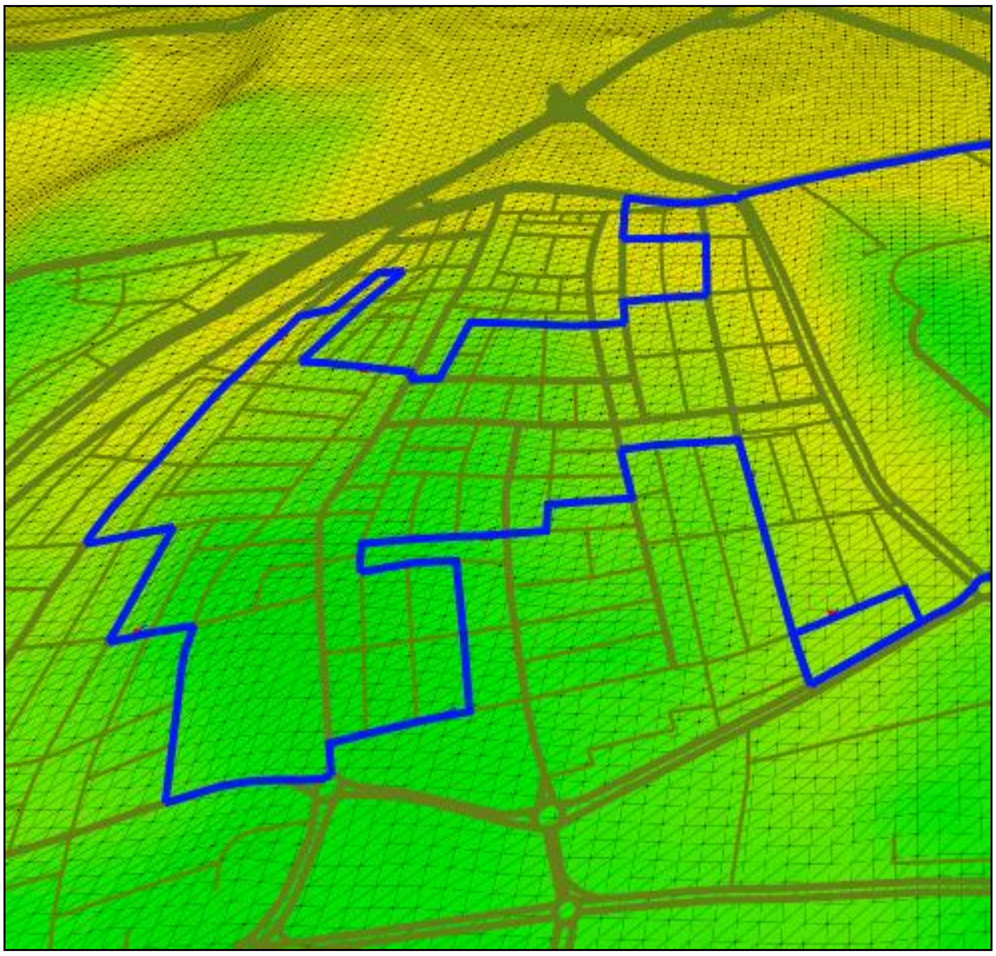


Figure 5.6 - Test Case 1 - A View of the Route Generated within the Study Area (Single Vehicle)

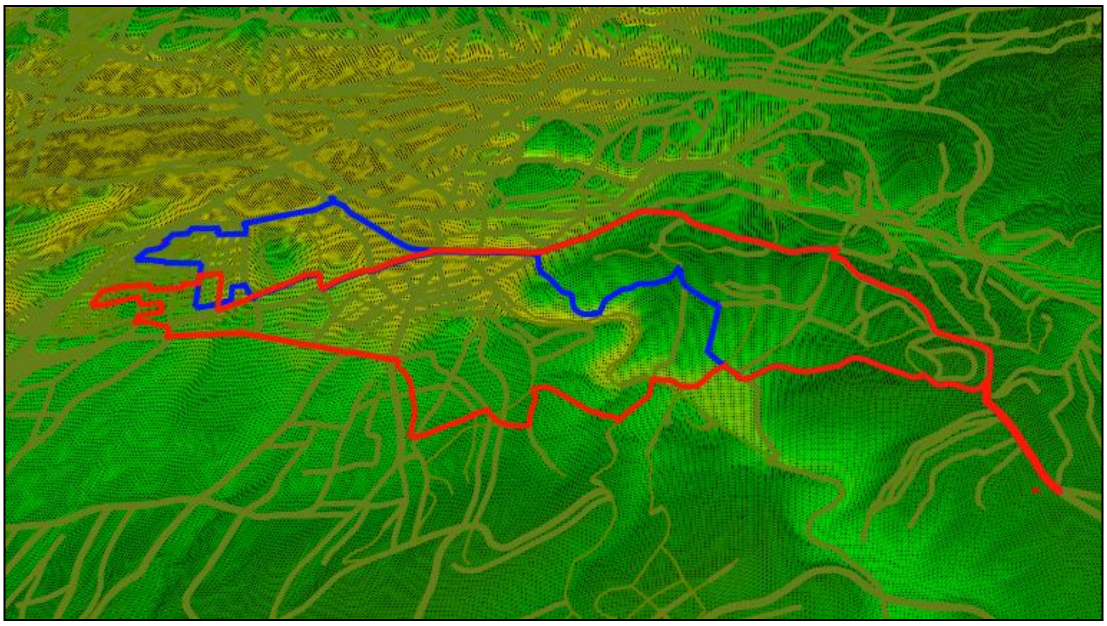


Figure 5.7 - Test Case 1 - A Full View of the Route Generated (2 Vehicles)

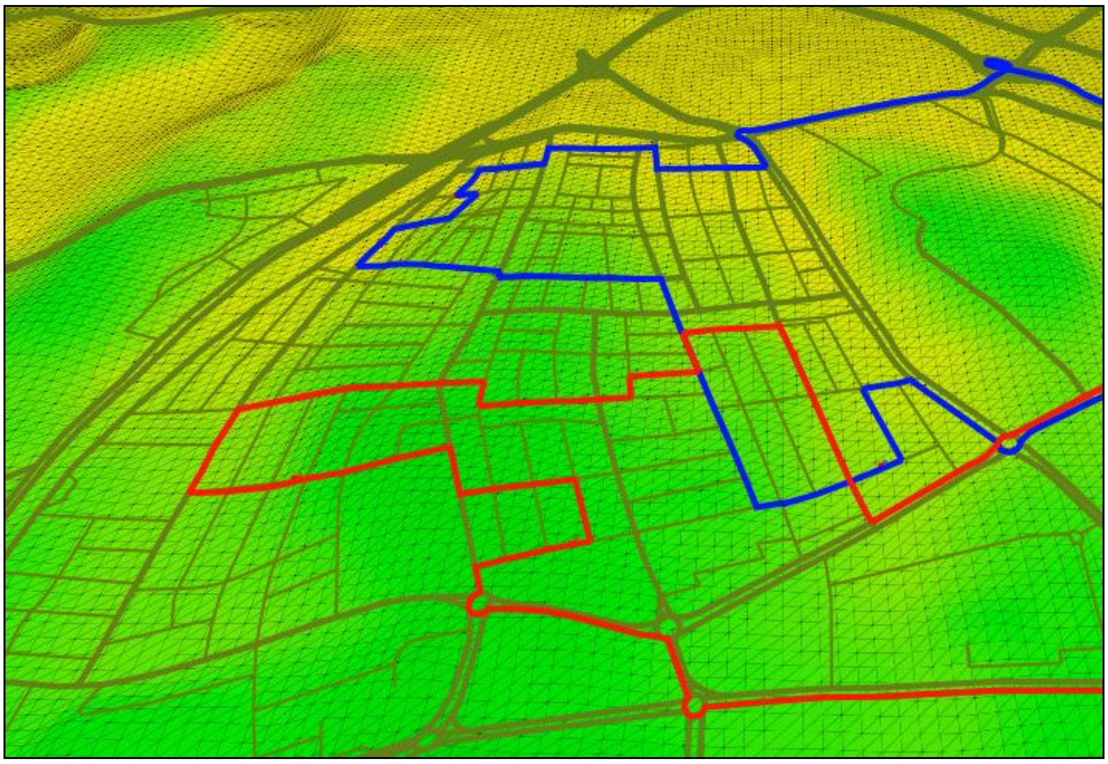


Figure 5.8 - Test Case 1 - A View of the Route Generated within the Study Area (2 Vehicles)

Table 5.1 – Test Case 1 - Total Route Lengths for Different Vehicle Counts

Number of Vehicles	Total Route Length (m)
1	30050.7
2	53889.7

For the second test case, 20 waste collection points are digitized in the study area with two clusters as shown in Figure 5.9.

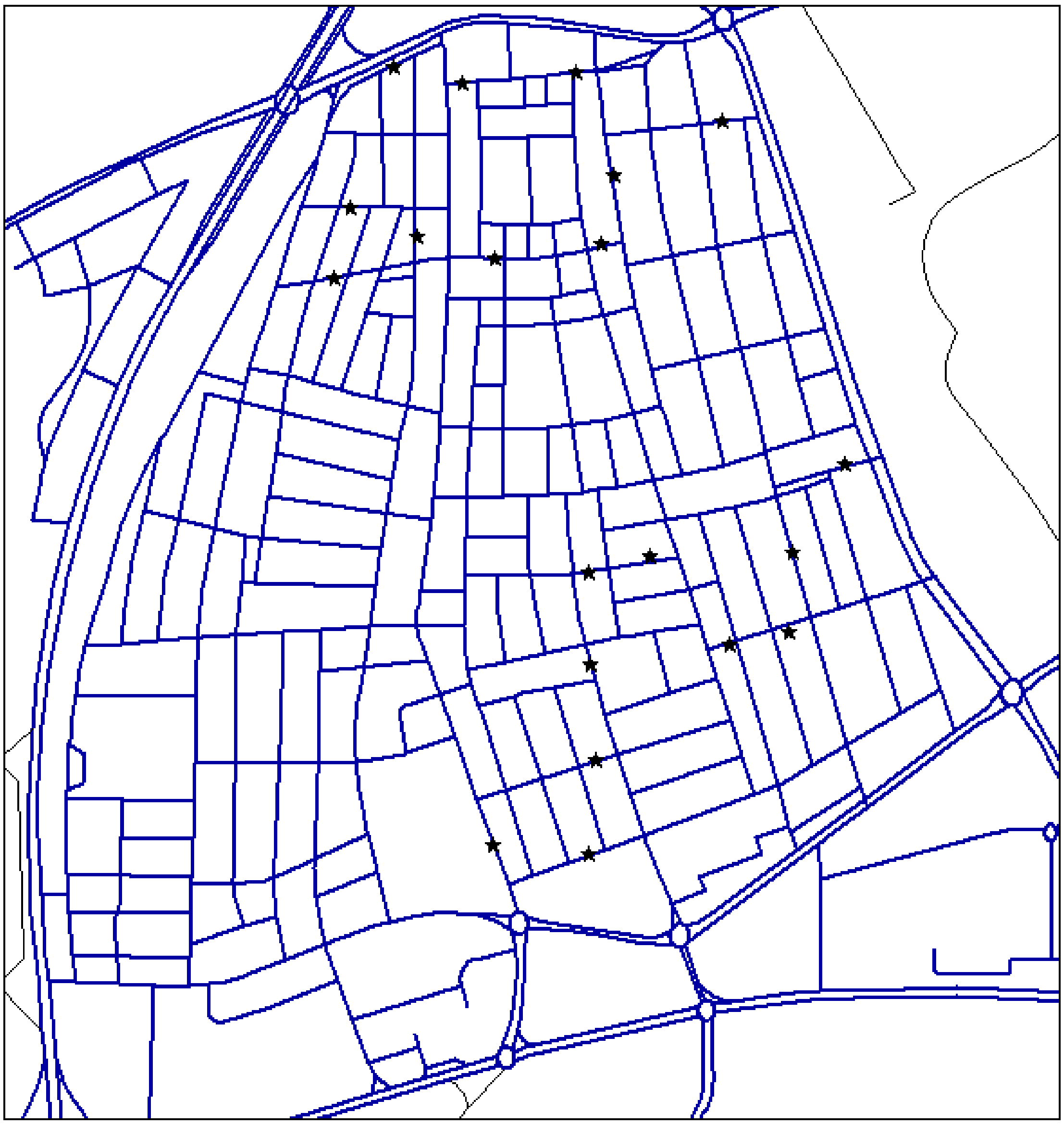


Figure 5.9 - – The Distribution of Waste Collection Locations in Test Case 2

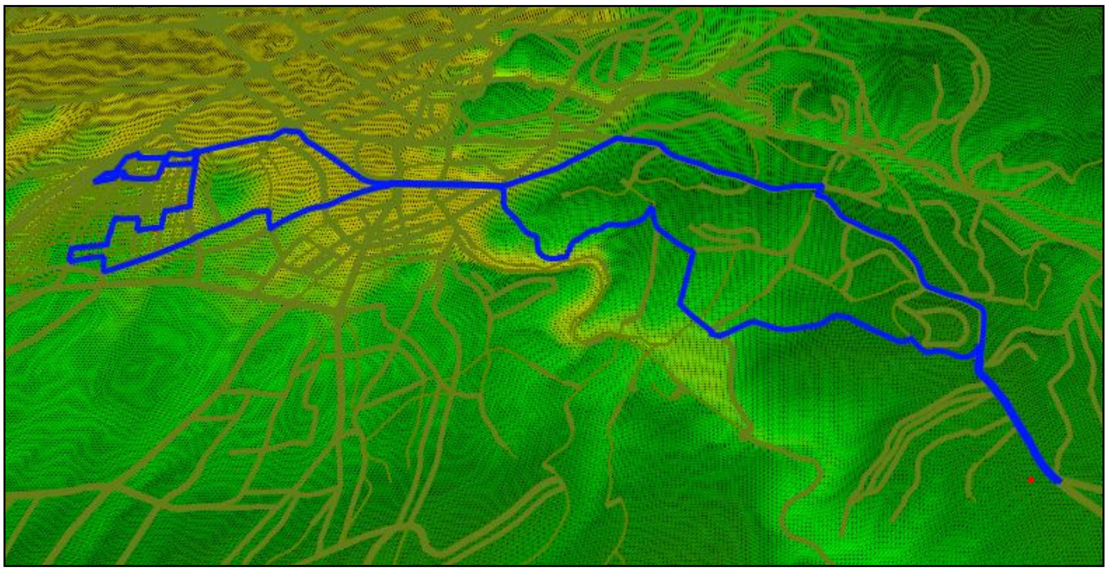


Figure 5.10 - Test Case 2 - A Full View of the Route Generated (Single Vehicle)

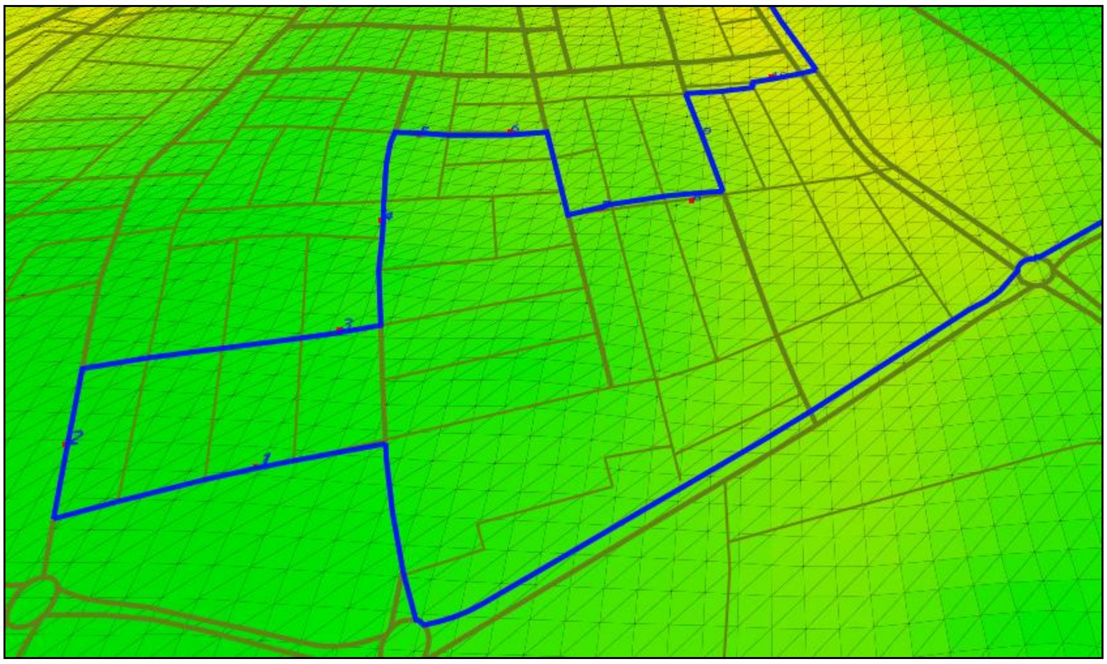


Figure 5.11 - Test Case 2 – A View of the Southern Part of the Route (Single Vehicle)

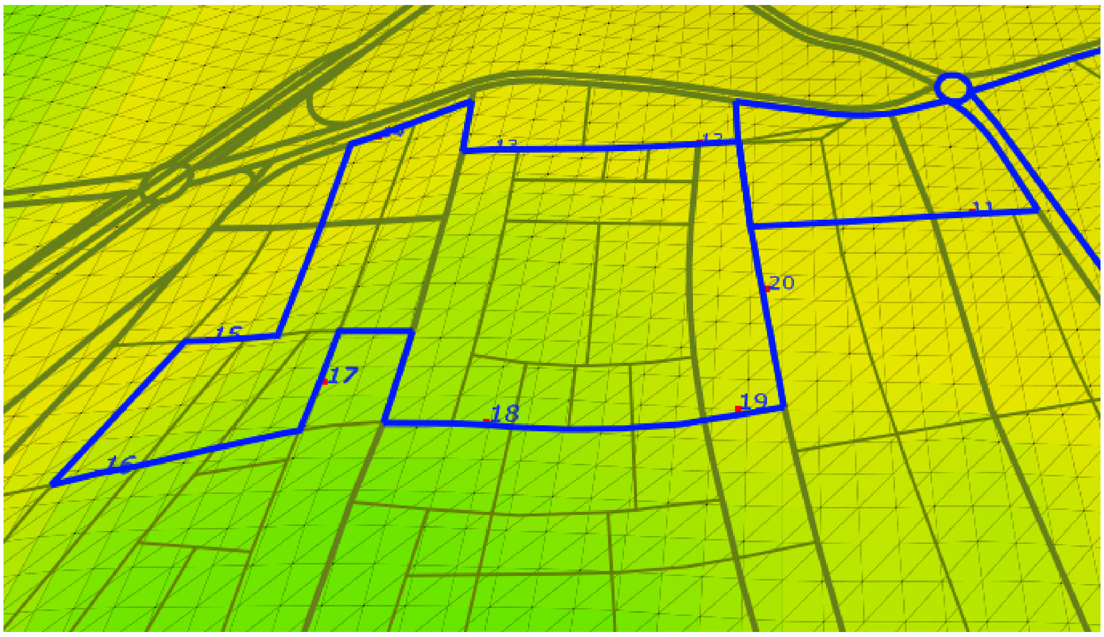


Figure 5.12 - Test Case 2 – A View of the Northern Part of the Route (Single Vehicle)

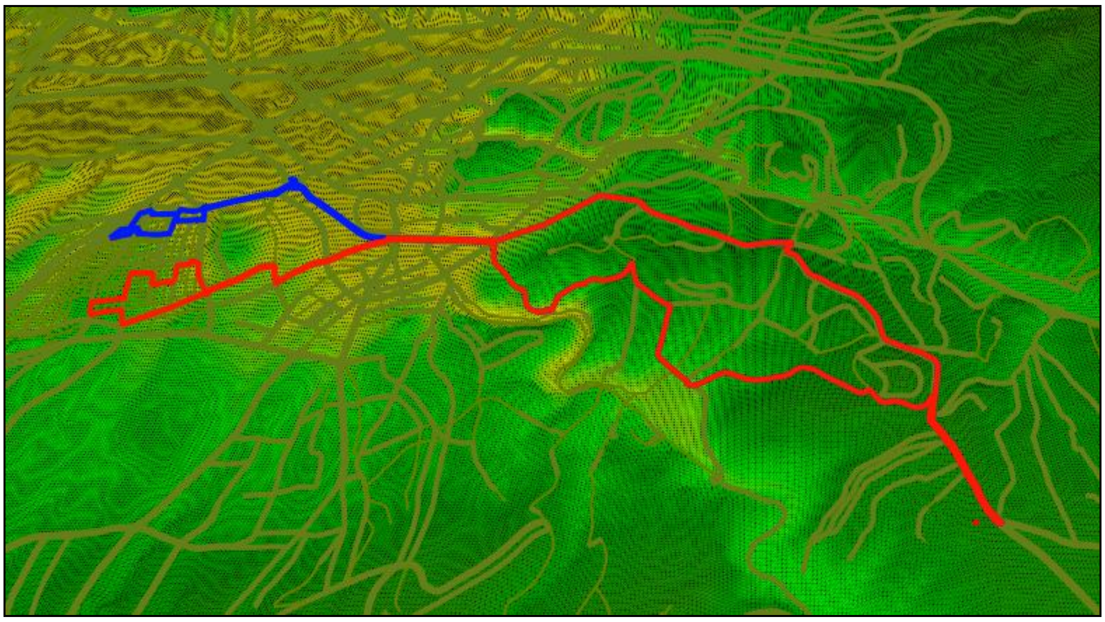


Figure 5.13 - Test Case 2 - A Full View of the Route Generated (2 Vehicles)

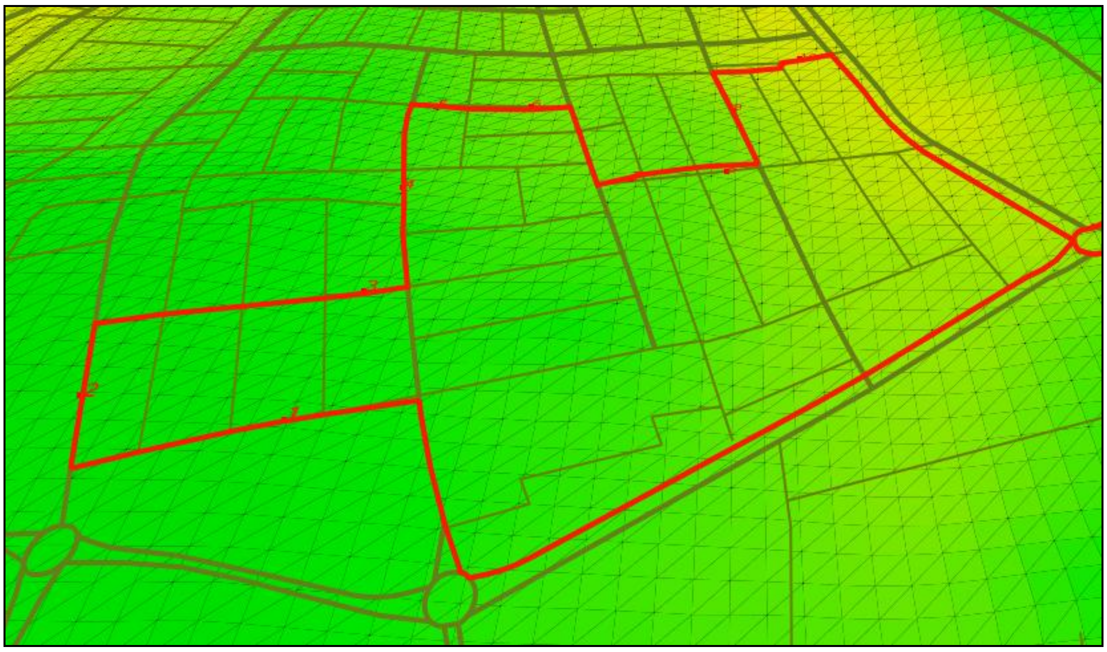


Figure 5.14 - Test Case 2 - A View of the Route Generated within the Study Area for the first vehicle (2 Vehicles)

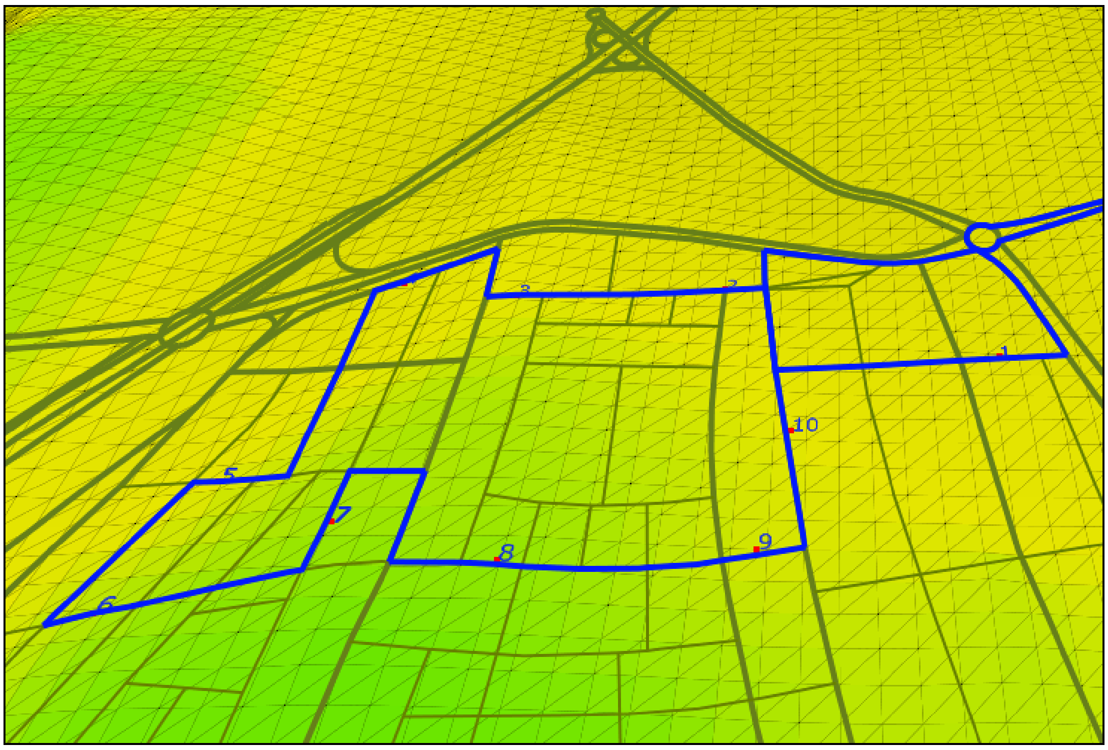


Figure 5.15 - Test Case 2 - A View of the Route Generated within the Study Area for the first vehicle (2 Vehicles)

Table 5.2 – Test Case 2 - Total Route Lengths for Different Vehicle Counts

Number of Vehicles	Total Route Length (m)
1	51840.3
2	29557.6

Regarding the test problems considered, it is clear that the paths generated are closed (i.e., start and end at the landfill). Each waste collection point is visited only once. Moreover, the generated routes comply with the road directions without any U-Turns. Furthermore, the route optimization algorithm succeeded in considering the clustered distribution of waste collection points in the Test Case 2. The route of each vehicle services one of the two clusters without any explicit algorithm to detect the clusters or any assignment procedure to assign the vehicles to points.

For the case study of Bahçelievler and Emek Neighbourhoods, the developed software is supposed to generate routes involving the landfill and 239 waste collection points for 3 trucks. The waste collection locations are uniformly distributed along the streets within the study area. Figure 5.16 shows the locations of the waste collection points. The test is made using two different optimization models: distance optimization mode and fuel consumption optimization mode. It is aimed to demonstrate the differences in routes arising from the model used.

In the first model, the objective is to minimize the total distance travelled by 3 trucks. Therefore the inclination of the roads and the instantaneous truck load is not taken into account and has no contribution on the penalty coefficient. Figure 5.17 and Figure 5.18 demonstrate the evolution of the routes.

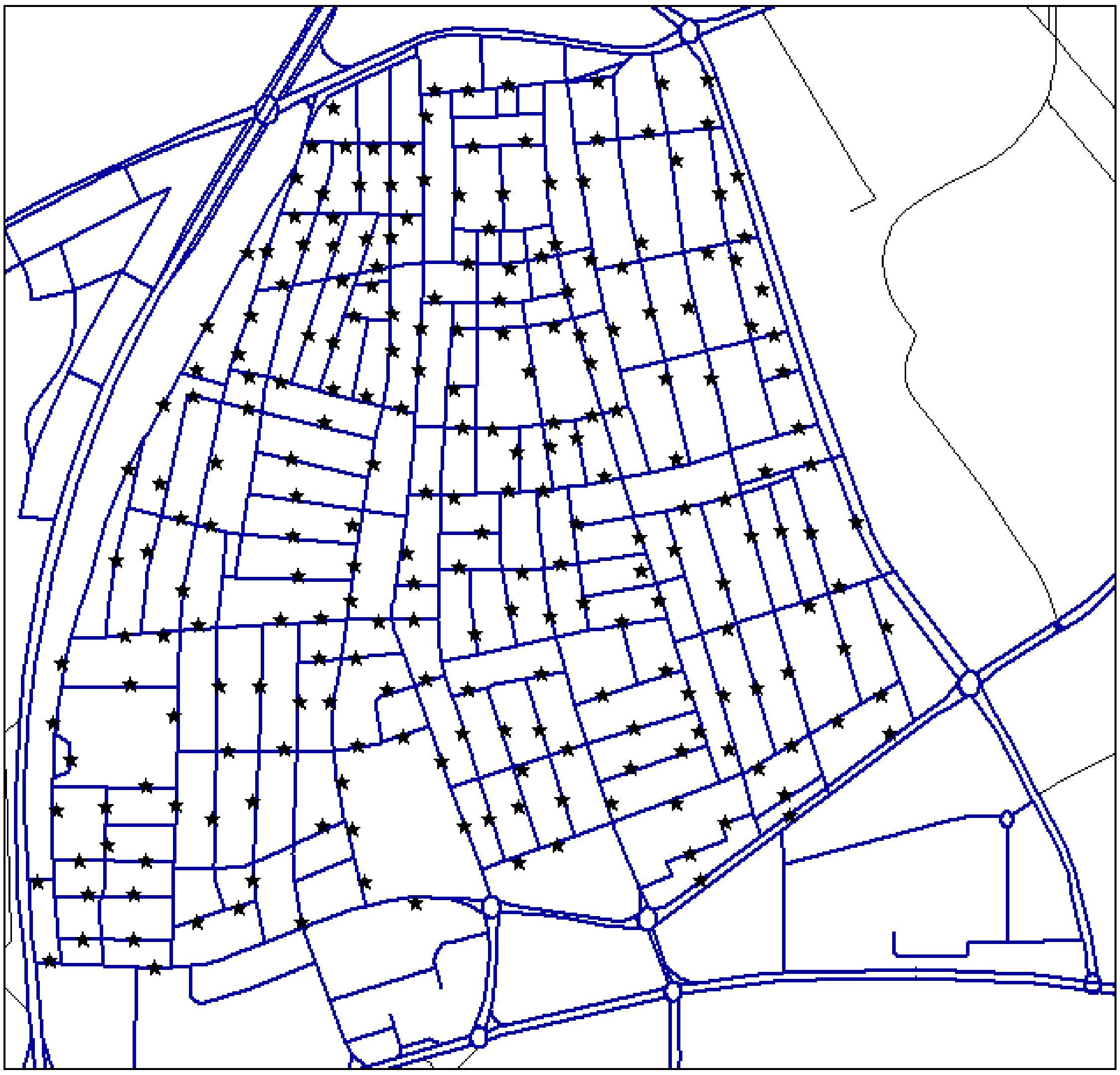


Figure 5.16 – The study area containing 239 Waste Collection Points

For the both the distance optimization model and the fuel consumption optimization model, the genetic algorithm is run 6 times. The lowest cost obtained in each model is marked and the resultant routes are illustrated.

Table 5.3 demonstrates the results of the total distance optimization model. The best route set has a total length of 119.9km, approximately.

Table 5.3 – Routes Generated by the Total Distance Optimization Model

Run #	Number of Generations	Time Passed(sec)	Route Length (m)
1	2045	301	121256.8
2	1193	192	122554.4
3	7356	324	121564.8
4	1658	213	120904.4
5	2856	245	121945.8
6	7616	504	119862.1

Figure 5.17 and Figure 5.18 illustrate the progress of the genetic algorithm for 6 different runs.

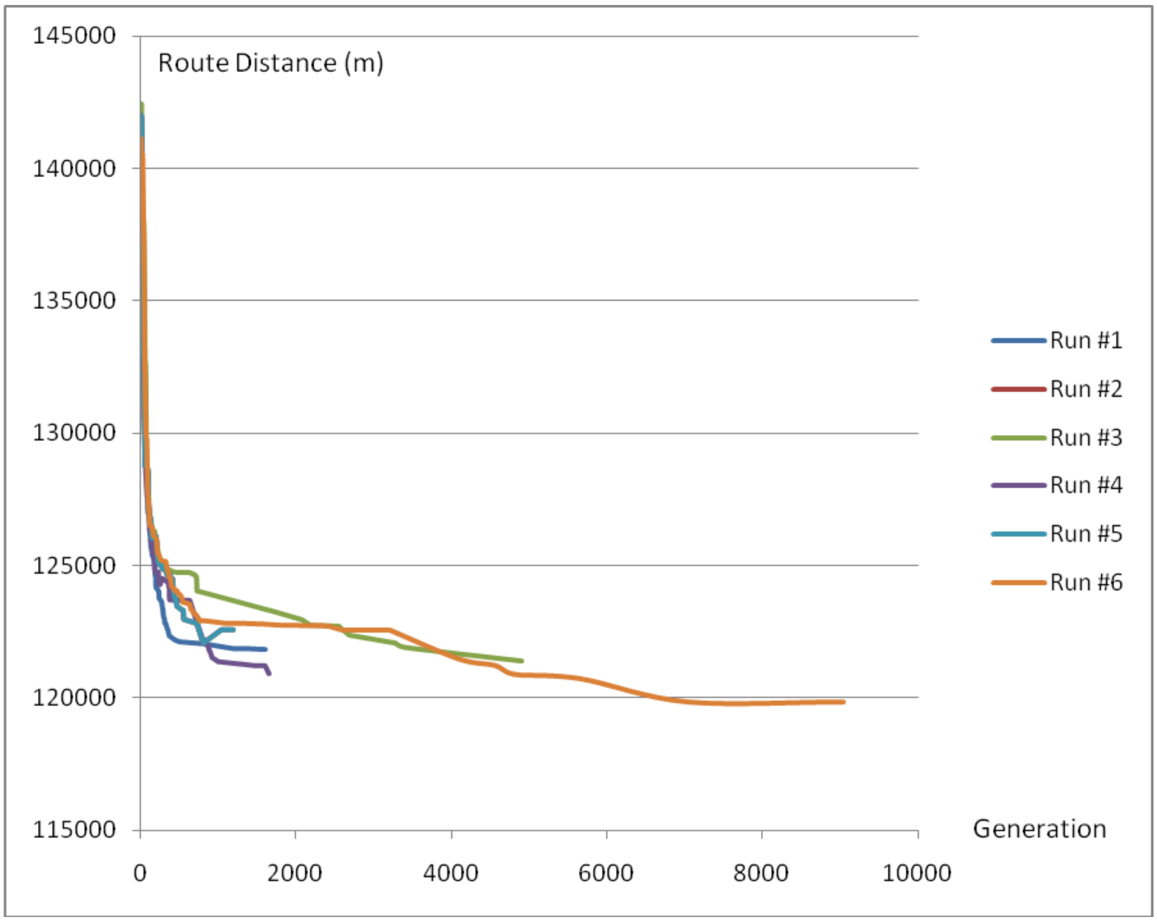


Figure 5.17 – Route Distance vs. Generation Graph for Distance Optimization Mode

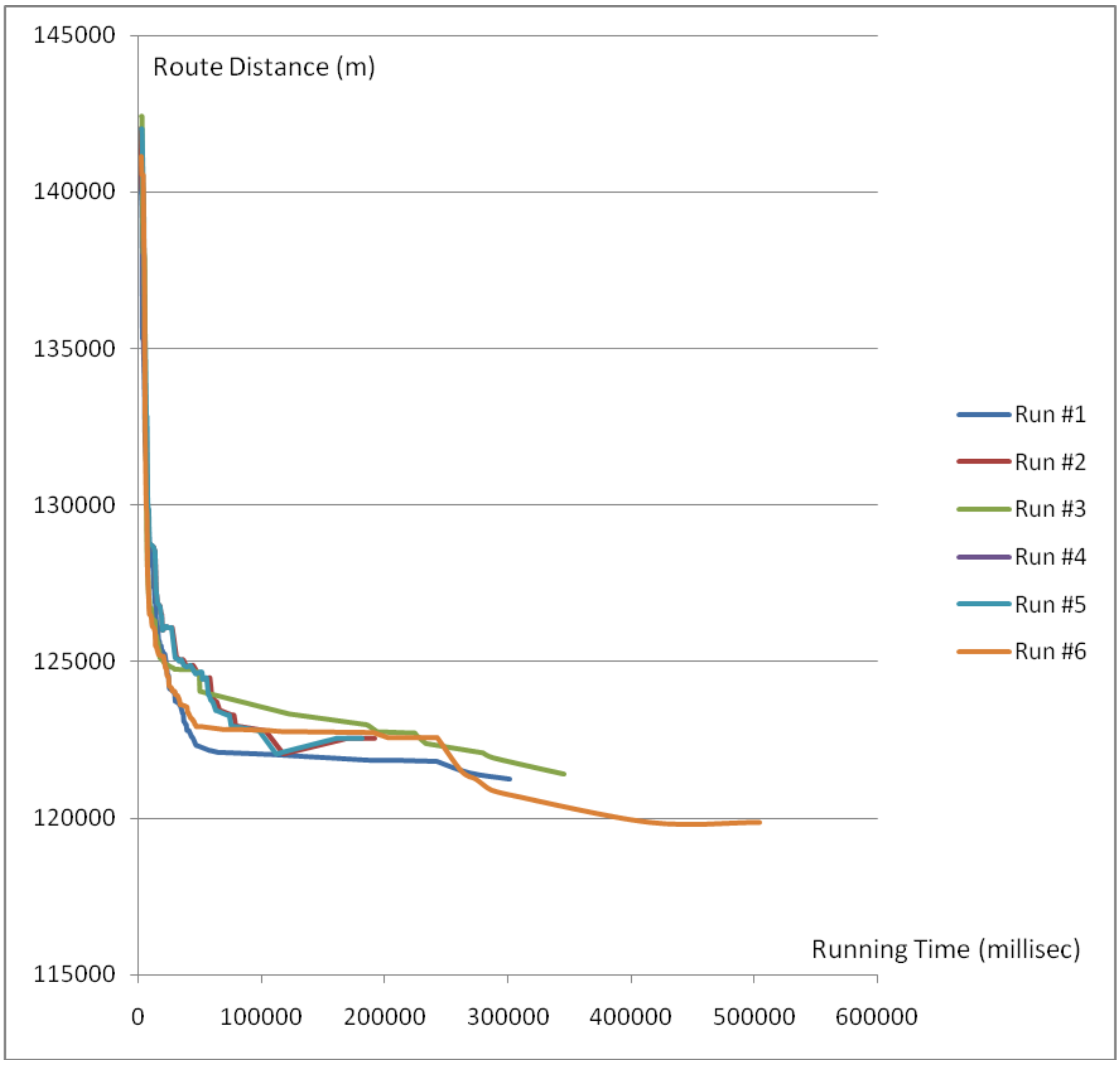


Figure 5.18 – Route Distance vs. Running Time Graph for Distance Optimization Mode

The software is run on an Intel Core2Duo Processor with 2GB of memory. The clock speed of each processor is 2.20GHz. During the test, all applications running in the background are closed so as to provide the same conditions for different runs.

The best solution is obtained in 761.6 generations within 504 seconds. The charts in Figure 5.17 and Figure 5.18 show that the solutions evolve very rapidly within the first 1000 generations. The evolution gradually slows down and the genetic algorithm is terminated after the termination criteria explained in Chapter 3 are satisfied. Figures Figure 5.19 throughFigure 5.23 illustrate different views of the routes generated by the distance optimization model.

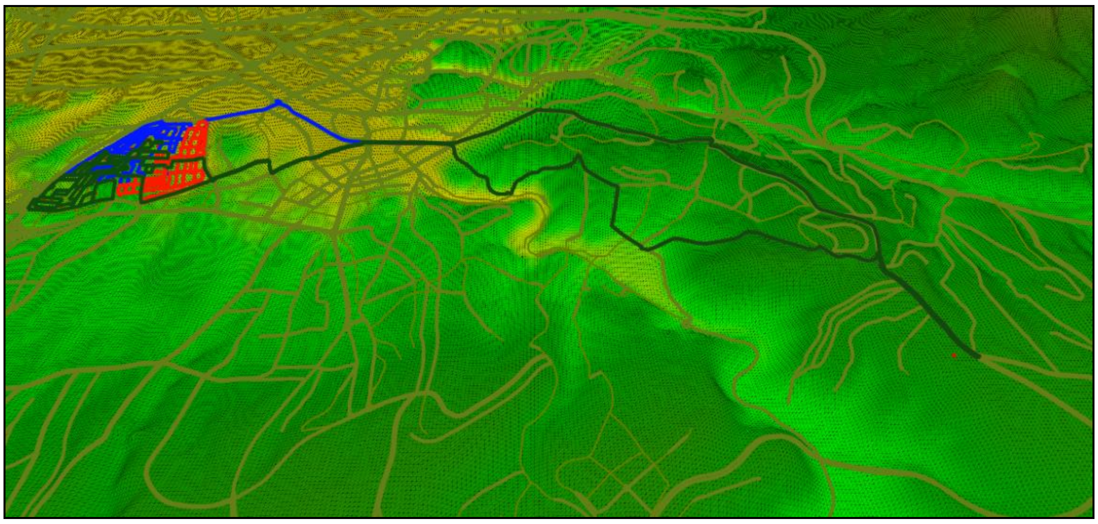


Figure 5.19 – Distance Optimization Model - A General Overview of Generated Routes

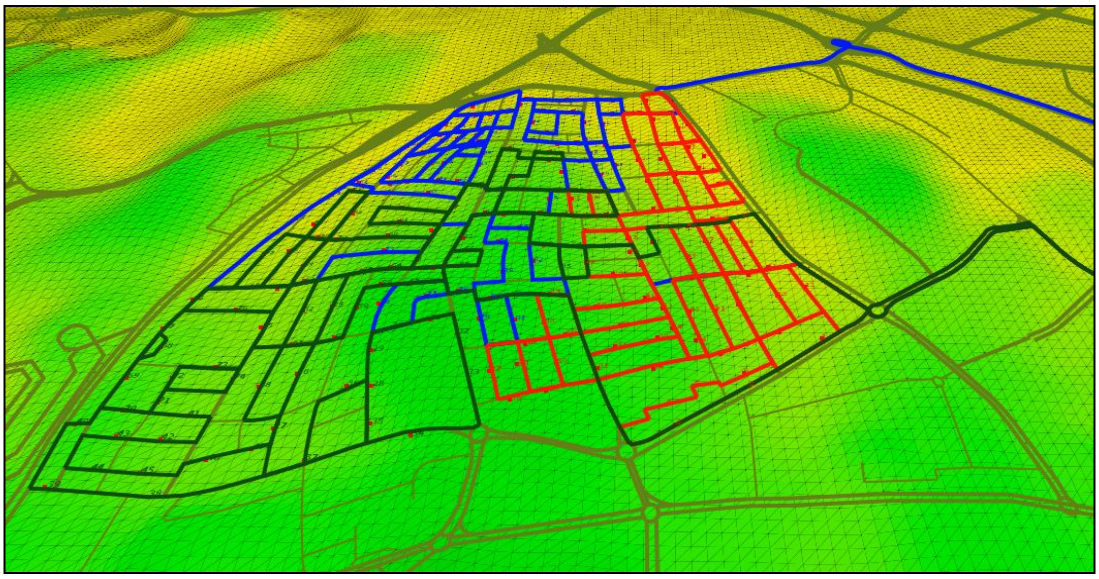


Figure 5.20 - Distance Optimization Model – Overview of Generated Routes in the Study Area

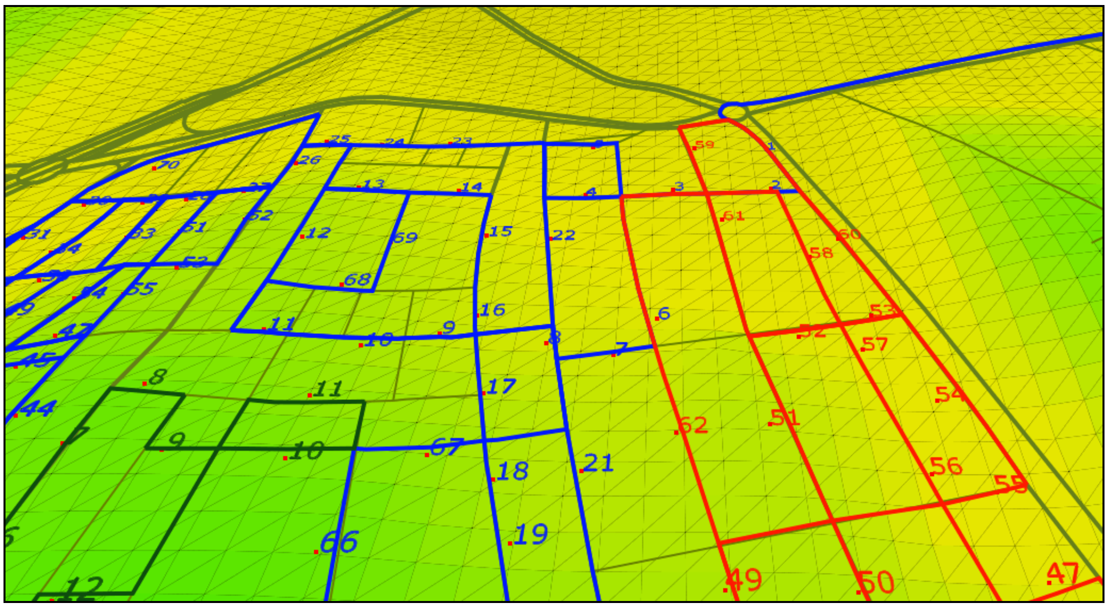


Figure 5.21 – Distance Optimization Model –Close-up view of North-East Bahçelievler

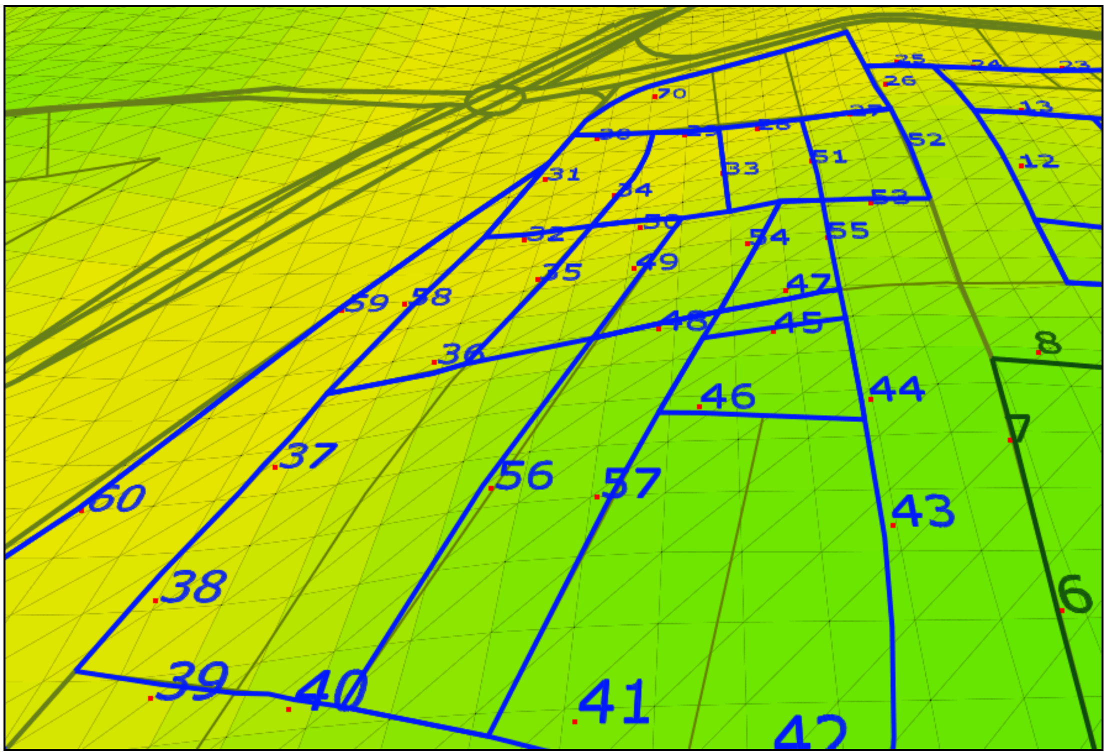


Figure 5.22 - Distance Optimization Model – Close-up view of Emek

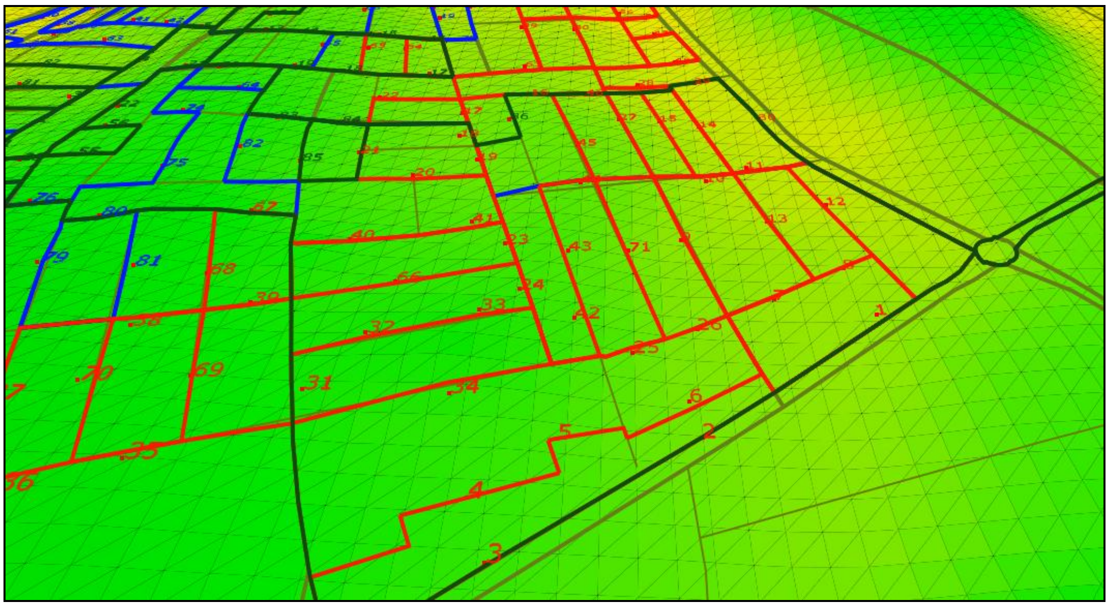


Figure 5.23 – Distance Optimization Model – Close-up view of Southern Bahçelievler

In the other optimization model which aims to minimize the total fuel consumption, a penalty coefficient is calculated for each edge of the routes generated based on the empirical model suggested by Tavares et al. (2009). According to the model, the fuel consumption depends on the road inclination and the instantaneous loading conditions of the vehicle as well as the distance travelled. Table 5.4 illustrates the roads generated by the fuel consumption optimization model.

Table 5.4 - Routes Generated by the Total Distance Optimization Model

Run #	Number of Generations	Time Passed(sec)	Route Length (m)
1	4356	388	121203.5
2	2169	176	122269.9
3	845	66	122269.9
4	2107	189	121207.8
5	2856	245	121945.8
6	2733	247	121856.6

The charts in Figures Figure 5.24 and Figure 5.25 demonstrate the evolution of the routes using the fuel consumption optimization model. It can be inferred from the charts that the solutions evolve rapidly in the first 500 generations and then the evolution gradually slows down. The algorithm is terminated when the termination criteria are satisfied.

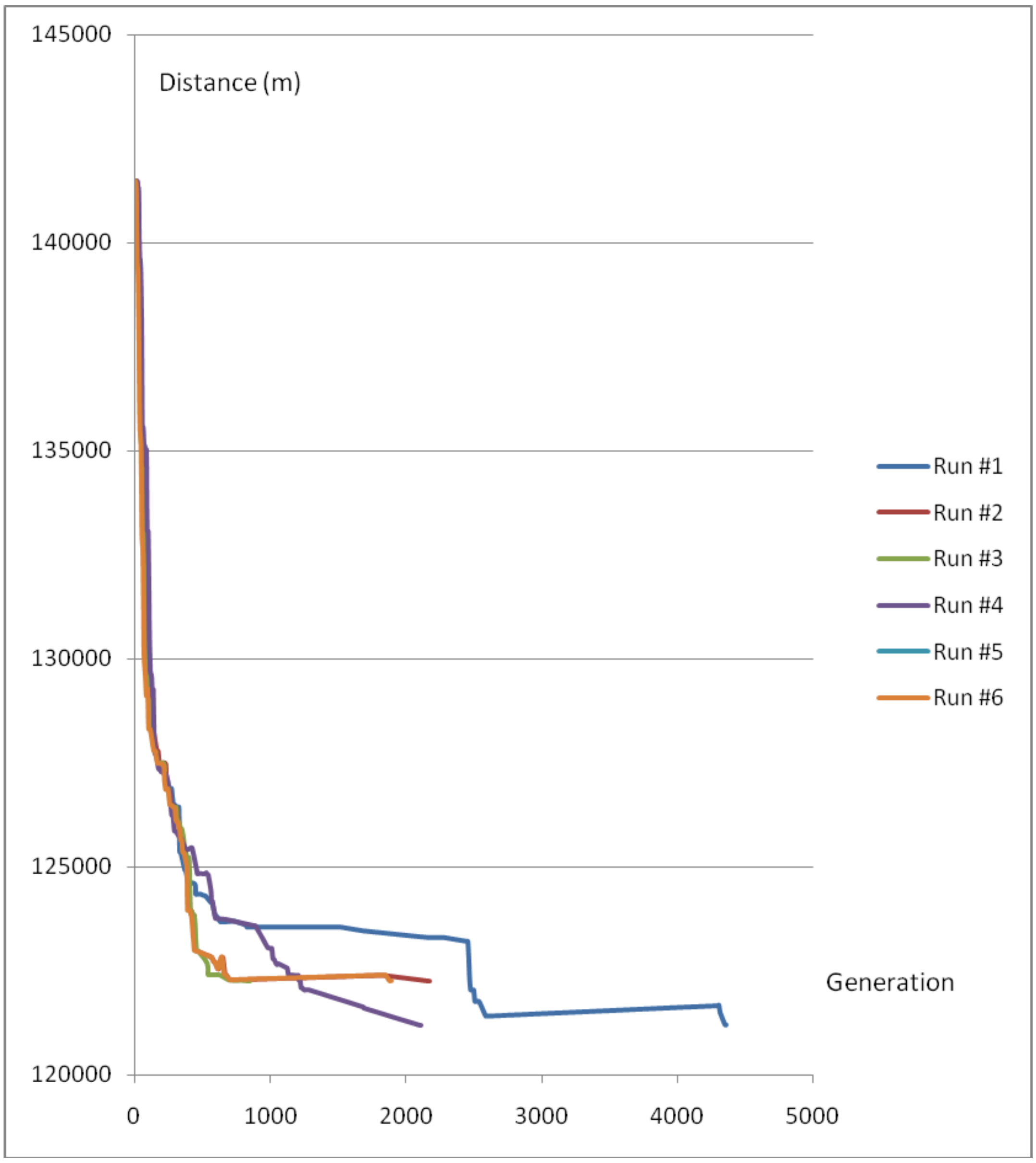


Figure 5.24 – Fuel Consumption Optimization Model - Distance vs Generation Graph

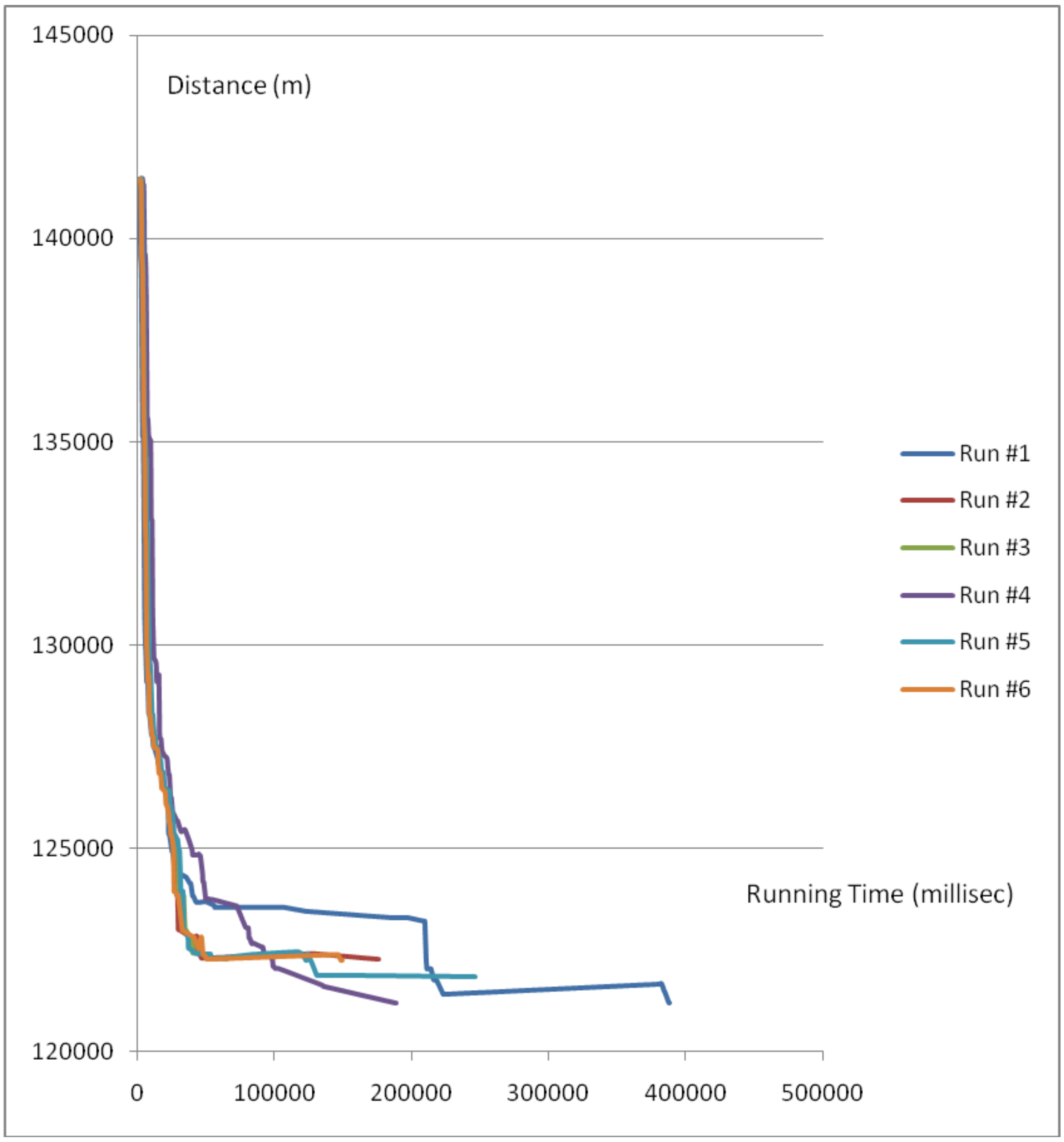


Figure 5.25 - Fuel Consumption Optimization Mode - Distance vs Running Time Graph

The Figures Figure 5.26 through Figure 5.30 illustrate the routes generated in different parts of the study area.

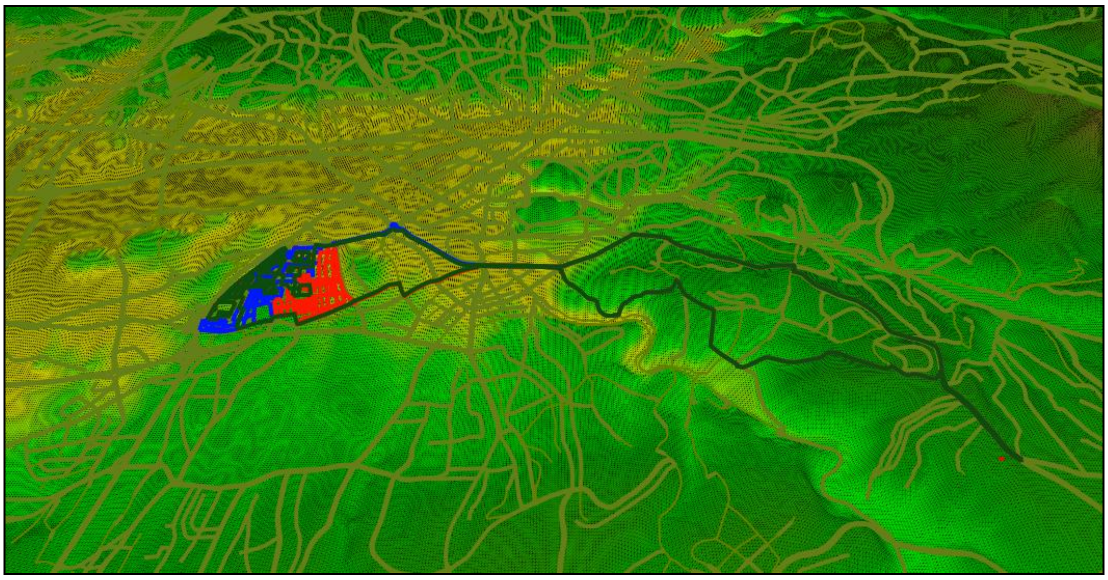


Figure 5.26 – Fuel Consumption Optimization Model - A General Overview of Generated Routes

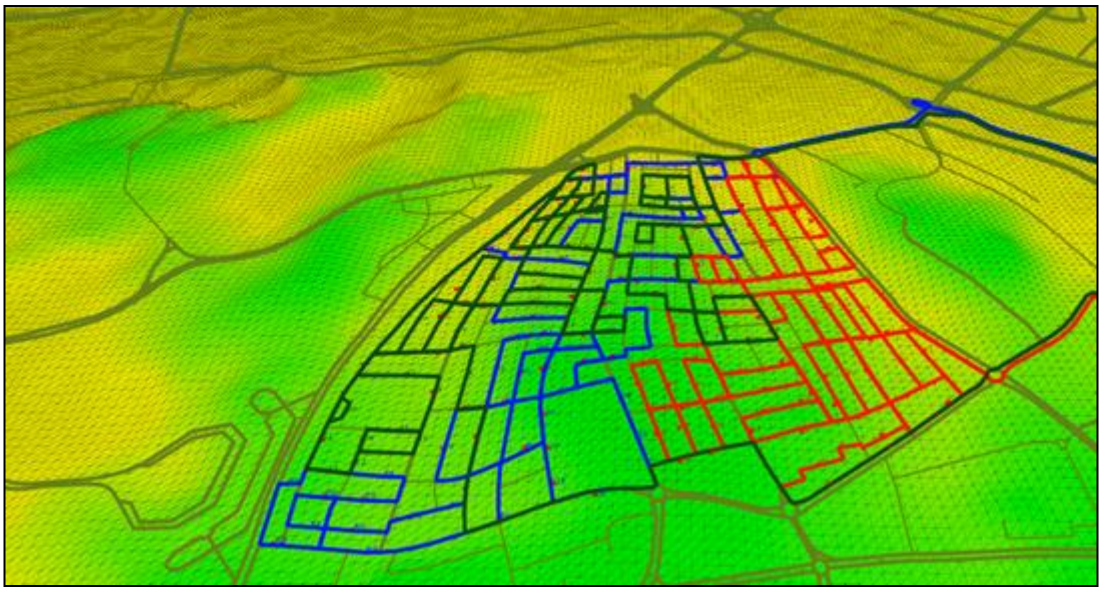


Figure 5.27 - Fuel Consumption Optimization Model - Overview of Generated Routes in the Study Area

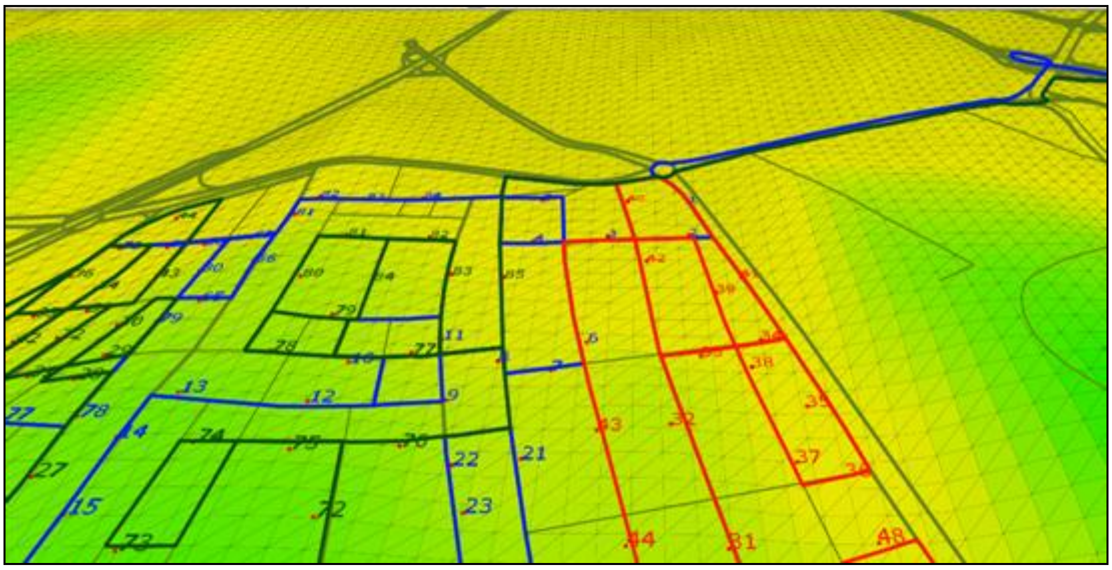


Figure 5.28 - Fuel Consumption Optimization Model - Close-up view of North-East Bahçelievler

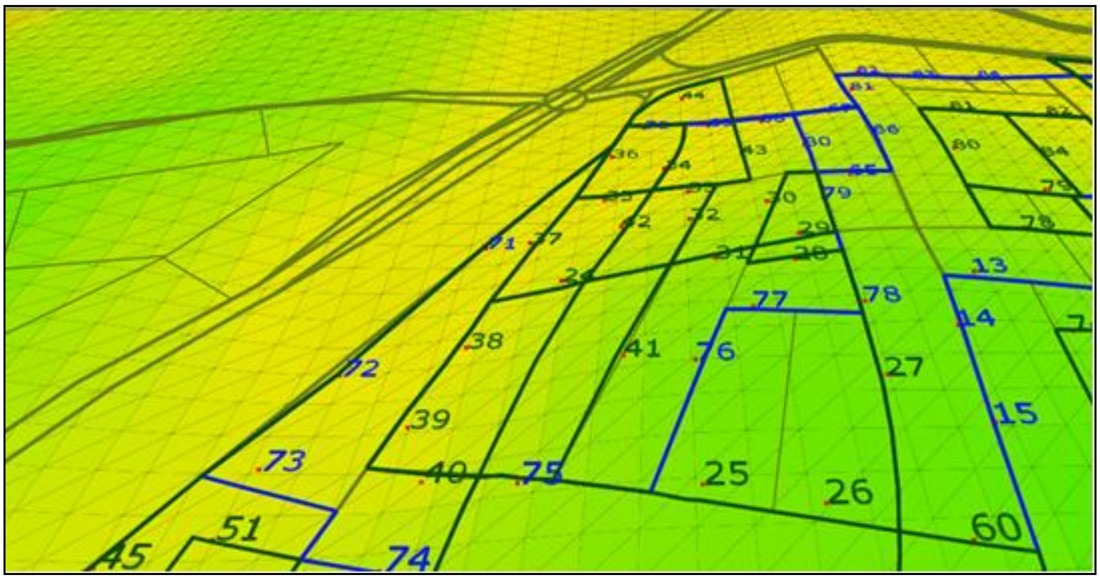


Figure 5.29 - Fuel Consumption Optimization Model – Close-up view of Emek

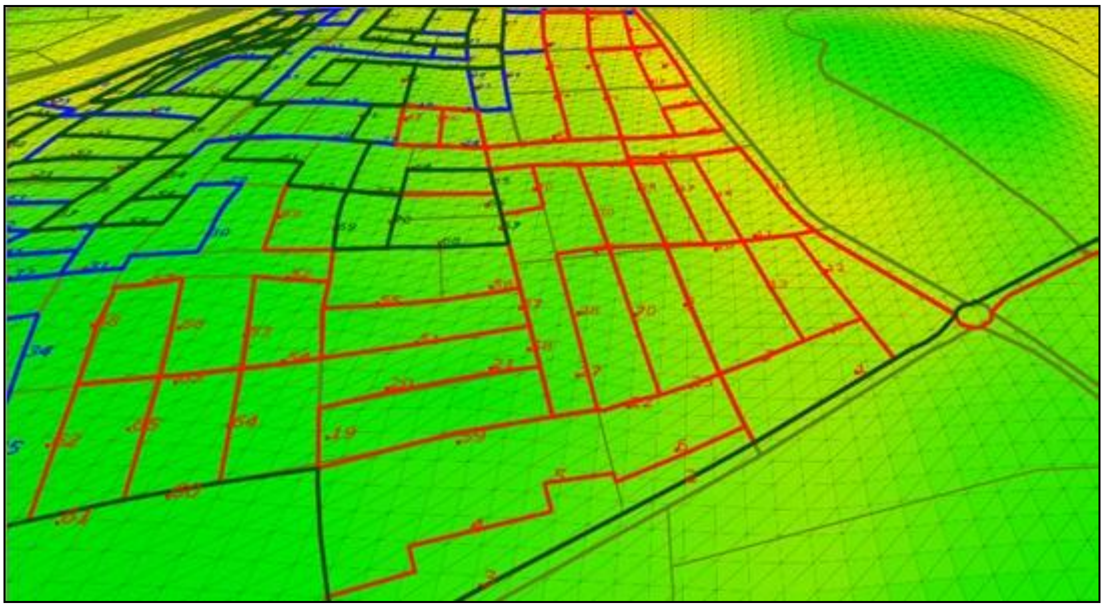


Figure 5.30 - Fuel Consumption Optimization Model - Close-up view of Southern Bahçelievler

Comparing the two models, it is expected that the model that minimizes fuel consumption handles the regions with steep slopes when the loading percentage of the waste collection truck is relatively low and flat regions when it is intensely loaded so as to minimize the fuel consumption. This behaviour can be seen in Figures Figure 5.31 and Figure 5.32.

Figure 5.31 demonstrates a route generated by the distance optimization model. Since the fuel consumption is not considered, the heavily loaded trucks can be routed along steep slopes. The truck represented by blue routes climb a very steep slope after having serviced 90 waste collection points.

On the other hand, Figure 5.32 illustrates a route generated in the same region using the fuel consumption optimization model. The collection route of the vehicle represented by green route starts from the steeply sloped region and the relatively flat areas are serviced later.

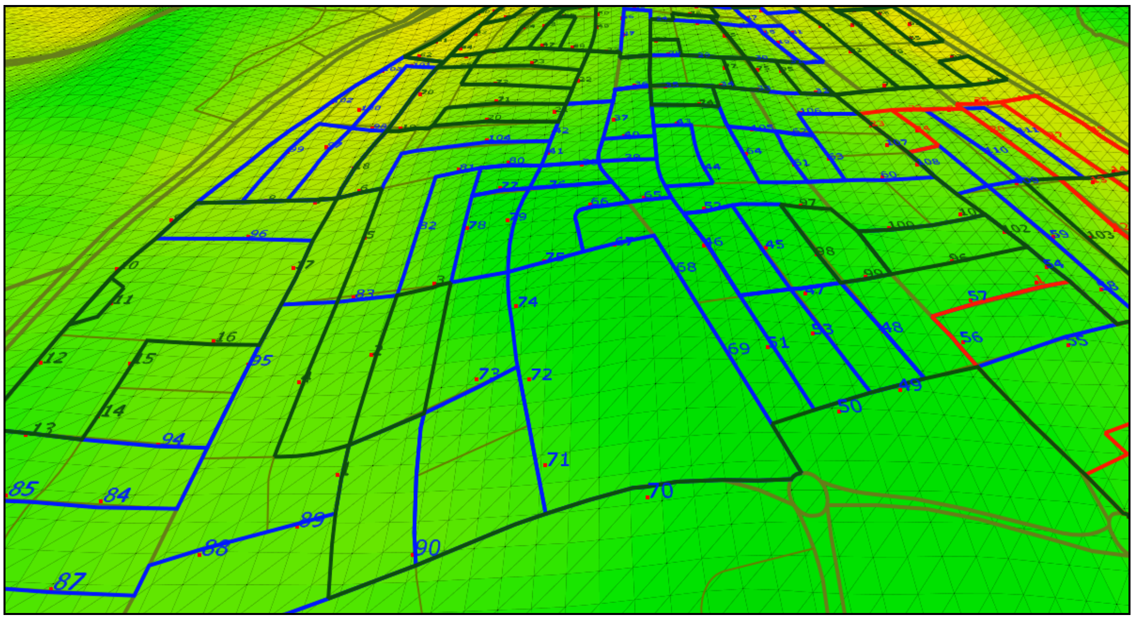


Figure 5.31 – Distance Optimization Model - A part of the route generated in a steeply sloping region in Emek

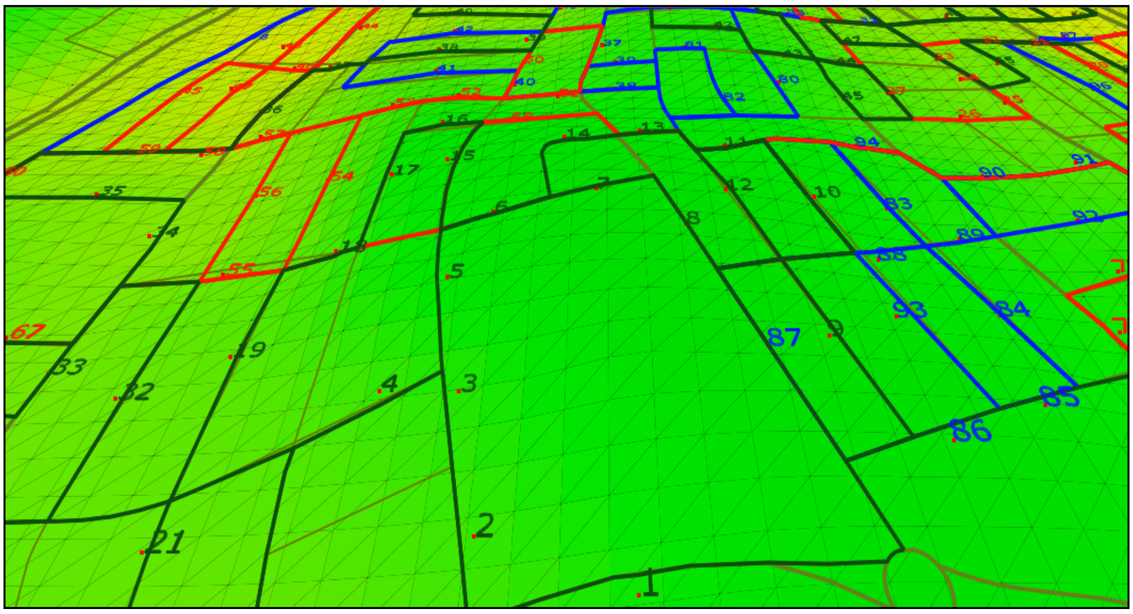


Figure 5.32 – Fuel Consumption Optimization Model - A part of the route generated in a steeply sloping region in Emek

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

Two different approaches are compared for route optimization: optimizing the total distance travelled and the fuel consumption. The objective of the first one is to minimize the total distance regardless of the effects of the road gradient and vehicle load. On the contrary, the second approach optimizes the fuel consumption estimated by an empirical model. It is shown that the two approaches generate different routes as the objective of the optimization changes.

The model is realistic as it considers the distances within the real road network using a digital elevation model (DEM). Moreover, the road directions are taken into consideration while generating routes. The model has a spatial constraint that avoids U-Turns, which is an essential element for waste collection routing.

The software provides two options for decision makers in selecting the waste collection routes: minimizing the total distance travelled and minimizing the total fuel consumption. This provides flexibility for decision makers so that the software can be used for different case studies rather than solid waste collection routing. Moreover, other fuel consumption models can be adapted easily by changing a single line of code.

As the developed software is run successfully for a moderately large number of points, it can directly be used for waste collection planning for larger applications. It is a trivial task to integrate new constraints to the model by modifying only the fitness function of the genetic algorithm. This is advantageous compared to exact algorithms as far as the amount of implementation work is considered.

The system is scalable in the sense that a high degree of parallelism is achieved in the tests conducted. Therefore, if the problem size is increased, the excess computational time can be compensated by increasing the number of cores in the computer on which the software is run.

For the future studies, this work can be applied to larger areas. Additional constraints such as minimization of left turns can be implemented to achieve a more goal-oriented routing system. Moreover, the software can be enhanced to support multiple landfills, which would add more complexity to the problem.

In addition to the recommendations for the possible future work-flow enhancements, it is also recommendable to modify the software so that it can operate in distributed networks rather than shared memory multi-core processors. This would significantly increase the scalability of the system. The pre-processing and optimization algorithms can be integrated to a tightly-coupled system so that application can run inside a GIS environment or within a GIS-based network.

REFERENCES

Al Jadaan O., Rajamani L., Rao C.R. (2008), "Improved selection operator for GA", *Journal of Theoretical and Applied Information Technology*, Vol. 4, No. 4, pp. 269 -277.

Altinkemer K., Gavish B. (1991), "Parallel Savings Based Heuristics for the Delivery Problem", *Operations Research*, Vol. 39, No. 3, pp. 456-469

Azi, N., Gendreau, M. and Potvin, J., (2010), "An exact algorithm for a single-vehicle routing problem with time windows and multiple use of vehicles.", *European Journal of Operational Research*. Vol. 202, No. 3, pp. 756-763.

Baker B.M., Ayechew M.A. (2003), "A genetic algorithm for the vehicle routing problem", *Computers and Operations Research*, Vol.30, No. 5, pp. 787-800

Bell J.E., McMullen P.R. (2004), "Ant colony optimization techniques for the vehicle routing problem", *Advanced Engineering Informatics* Vol.18, No.1, pp.41–48

Bin Y., Zhong-Zhen Y., Baohzen Y. (2009), "An improved ant colony optimization for vehicle routing problem", *European Journal of Operational Research*, Vol.196, No. 1, pp. 171-176

Bjarnadóttir Á.S. (2004), "Solving the Vehicle Routing Problem with Genetic Algorithms" (Master's Thesis), *Informatics and Mathematical Modelling*, Technical University of Denmark

Clarke K.C. (1986), "Advances in Geographic Information Systems, Computers, Environment and Urban Systems", Vol. 10, No. 3-4, pp. 175-184

Clarke, G., Wright, J.V. (1964), "Scheduling of vehicles from a central depot to a number of delivery points", *Operations Research*, Vol. 12, No.1, pp. 568-581

Davis, L. (1985), "Applying Adaptive Algorithms to Epistatic Domains", in Proceedings of the *International Joint Conference on Artificial Intelligence*", pp.162-164.

Donati A.V., Montemanni R., Casagrande N., Rizolli A.E., Gambardella L.M. (2008), "Time Dependent Vehicle Routing Problem with a Multi Ant Colony System", European Journal of Operational Research, Vol. 185, No. 3, pp. 1174-1191

Dorigo M., Gambardella L.M. (1997), "Ant colony system: a cooperative learning approach to the traveling salesman problem*", IEEE Transactions on Evolutionary Computation*, Vol.1, No.1, pp.53-66.

Dorigo M., Maniezzo V., Colorni A. (1996), "Ant system: optimization by a colong of cooperating agents", *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Vol.26, No.1,1996, pp.29-41

El-Mihoub T.A., Hopgood A.A., Nolle L., Battersby A. (2006), "Hybrid Genetic Algorithms: A Review", *Engineering Letters*, ISSN: 1816-0948, 13:2, EL_13_2_11

Fisher M.L., (1994), "Optimal solution of vehicle routing problems using Minimum K-trees", *Operations Research*, Vol. 42, No 4, pp. 626-642

Fisher M.L., Jaikumar R. (1981), "A generalized assignment heuristic for vehicle routing", *Networks*, Vol.11, No.2, pp.109-124

Fisher M.L., Jörnsten K.O., Madsen B.G.O., (1997), “Vehicle Routing with Time Windows: Two optimization Algorithms”, *Operations Research,* Vol.45, No. 3, pp.488-492

Ghose, M.K., Dikshit, A.K., Sharma, S.K., (2006), “A GIS based transportation model for solid waste disposal – a case study on Asansol municipality.”, *Waste Management*, Vol. 26, No.11, pp.1287–1293

Gillett B.E., Miller L.R. (1974) "A Heuristic Algorithm for the Vehicle-Dispatch Problem", *Operations Research*, Vol. 22, No. 2, pp. 340-349

Gillett, B.E., Miller, L.R. (1974), “A heuristic algorithm for the vehicle dispatch problem”, *Operations Research*, Vol. 22, No.2, pp. 240–349.

Goldbeg D.E. (1989), “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA

Goldberg D.E., Miller B.L. (1995), “Genetic Algorithms, Tournament Selection, and the Effects of Noise”, *IlliGAL Report,* No. 95006

Goldberg, D.E., Lingle, R. (1985), “Alkies, Loci, and the TSP”, in Proceedings of the *First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, pp.154-159.

Ho, W., Ho, G.T.S., Ji, P., Lau, H.C.W. (2008), “A hybrid genetic algorithm for the multi-depot vehicle routing problem”, *Engineering Applications of Artificial Intelligence,* Vol. 21, pp. 548-557

Holland J. (1975), “Adaptation in Natural and Artificial Systems”, *University of Michigan Press,* Ann Arbor, MI. 9

Keenan P. (2008), “Modelling Vehicle Routing in GIS”, *Operational Research,* Springer Berlin / Heidelberg, Vol 8, No 3, pp.201-218

Kinderwater, G.A.P., Savelsbergh, M.W.P. (1997), “Vehicle Routing: Handling Edge Exchanges” in *Local Search in Combinatorial Optimization*. Wiley, Chichester.

Kirkpatrick S., Gelatt C. D., Vecchi M.P. (1983) “Optimization by Simulated Annealing”, *Science. New Series*, Vol.220, No. 4598, pp. 671–680

Lin S., Kernighan BW. (1973), "An effective heuristic algorithm for the TSP.", *Operations Research*, Vol.21, No.1, pp.498–516

Machado P., Tavares J., Pereira F.B., Costa E. (2002), “Vehicle Routing Problem: Doing It The Evolutionary Way”, *Proceedings of the Genetic and Evolutionary Computation Conference*, p.690, July 09-13

Malik W., Rathinam S., Darbha S. (2007), “An approximation algorithm for a symmetric Generalized Multiple Depot, Multiple Travelling Salesman Problem”, *Operation Research*, Letters Vol.35, No.6, pp.747-753

Michalewicz, Z., 1996, “Genetic Algorithms + Data Structures = Evolution Programs”, Springer–Verlag, Berlin, pp.208-237

Mosheiov G. (1998), “Vehicle routing with pick-up and delivery: tour-partitioning heuristics”, *Computers & Industrial Engineering*, Vol. 34, No. 3, pp. 669-684

Nazif H., Lee L.S. (2010), “Optimized Crossover Genetic Algorithm for Vehicle Routing Problem with Time Windows”, *American Journal of Applied Sciences*, Vol.7, No.1, ISSN 1546-9223

Nemerow, N.L., Agardy F.J. (2008), “Environmental engineering: Environmental health and safety for municipal infrastructure, land use and planning, and industry”, John Wiley and Sons, 558p.

Ntziachristos, L., Samaras, Z. (2000), “COPERT III – Computer Programme to Calculate Emissions from Road Transport, Methodology and Emission Factors (Ver. 2.1)”, *EEA*, Copenhagen

Nuortio T., Kytöjoki J., Bräysy O. (2006), “Improved route planning and scheduling of waste collection and transport”, *Expert Systems with Applications*, Vol. 30, No.2, pp. 223–232

Rizzoli, A.E., Oliverio F., Montemanni R., Gambardella L.M. (2004), “Ant Colony Optimization for vehicle routing problems: from theory to applications”, *Technical Report IDSIA-15-04*, Instituto Dalle Molle di Studi sull’Intelligenza Artificiale, Lugano, Switzerland

Ryan, D.M., Hjorring, C. and Glover, F. (1993) "Extensions of the petal method for vehicle routing", *Journal of the Operational Research Society,* Vol. 44, No.1, pp.289–296.

Sengoku H., Yoshihara B. (1998), “A Fast TSP Solver Using GA on JAVA”, *Third International Symposium on Artificial Life and Robotics* (AROB III'98)

Shengjun X., Shaoyong G., Dongling B. (2008), “The Analysis and Research of Parallel Genetic Algorithm”, *Wireless Communications, Networking and Mobile Computing WiCOM '08. 4th International Conference*, 12-14 October 2008, Dalian.

Sumichras R.T., Markham I.S. (1995), “A heuristic and lower bound for a multi-depot routing problem”, *Computers & Operations Research*, Vol. 22, No. 10, pp. 1047-1056

Tan K.C., Lee L.H., Ou K. (2001), “Artificial intelligence heuristics in solving vehicle routing problems”, *Engineering Applications of Artificial Intelligence*, Vol. 14, No.1, pp. 825–837

Tarantilis, C.D., Ioannou, G., Prastacos G. (2005), “Advanced vehicle routing algorithms for complex operations management problems”, *Journal of Food Engineering*, Vol.70, No.3, pp. 455-471

Tavares G., Zsigraiova Z., Semiao V., Carvalho M.G. (2009), “Optimisation of MSW collection routes for minimum fuel consumption using 3D GIS modelling”, *Waste Management,* Vol.29, No.3, pp. 1176–1185

Tchobanoglous G., Thiesen H., Vigh S.A. (1993), “Integrated Solid Waste Management – Engineering Principles and Management Issues”, McGraw-Hill International Editions, 978 p.

Thompson, P.M., Psaraftis, H.N. (1993) “Cyclic transfer algorithms for the multi-vehicle routing and scheduling problems”, *Operations Research,* Vol. 41, No.5, pp. 935–946

Toth, P., Vigo, D. (2000), “The Vehicle Routing Problem, an Overview of Vehicle Routing Problems”, *Society for Industrial and Applied Mathematics(SIAM),* Philadelphia, USA, pp.1-26

Van Breedam, A. (1994), "An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle-Related, Customer-Related and Time-Related Constraints", Ph.D. Dissertation, University of Antwerp

Vesilind P.A., Worrel W.A., Reinhart D.R. (2001), "Solid Waste Engineering", Nelson Engineering, ISBN: 8131504107, 448 p.

Whitley, D., Starkweather, T., and Fuquay, D'A. (1989), “Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator”, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San ma Mateo, CA, 1989.

Yoshikawa M., Nagura T. (2009), “Adaptive Ant Colony optimization Considering Intensification and Diversification”, *Proceedings of the International MultiConference of Engineers and Computer Scientists,* Vol.1, March 18-20, 2009, Hong Kong

ONLINE REFERENCES

ESRI – ESRI Shape File Technical Documentation, Retrieved December 2010, from http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

Golden Software – Surfer, A Powerful Contouring, Gridding, and Surface Mapping Package for Scientists and Engineers, Retreived October 2010, from http://www.goldensoftware.com/products/surfer/surfer.shtml

OpenGL, The Industry's Foundation for High Performance Graphics, Retreived December 2010, http://www.opengl.org/

Parallel Programming Lecture 14 Computer performance, Retreived January 2011, http://lib.bioinfo.pl/files/courses/pdfcache/lecture_461.pdf

SRTM – Shuttle Radar Topography Mission, Retreived October 2010, from http://www2.jpl.nasa.gov/srtm/

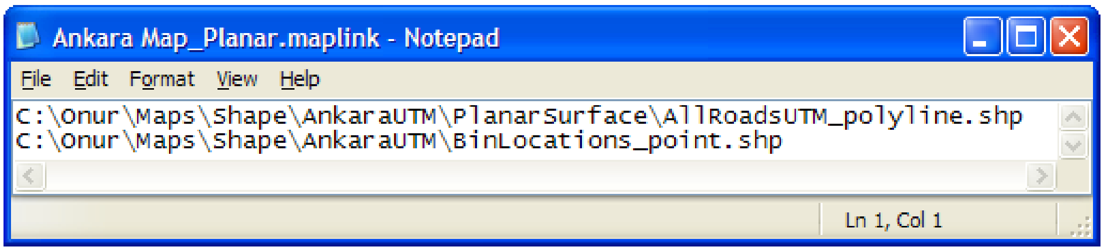
TSPLIB - Asymmetric traveling salesman problem (ATSP), Retrieved October 2010, from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

APPENDIX A

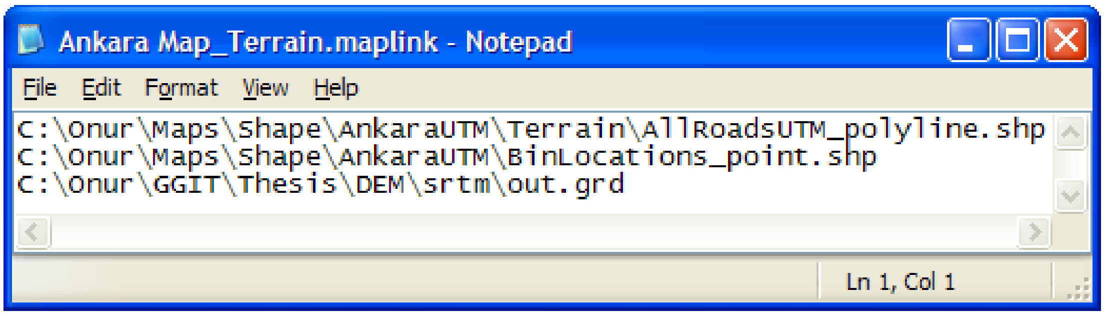
SOFTWARE MANUAL

The developed software requires two shape files as input: one for the road network and the other for garbage collection points. A Surfer Grid file can optionally be supplied for 3D terrain view.

The application is designed to open files of maplink extension, which can easily be created using notepad. The file includes 3 lines containing full paths to road network shape file, garbage collection point shape file and the altitude grid file.


Ankara Map_Planar.maplink - Notepad
File Edit Format View Help
Ln 1, Col 1

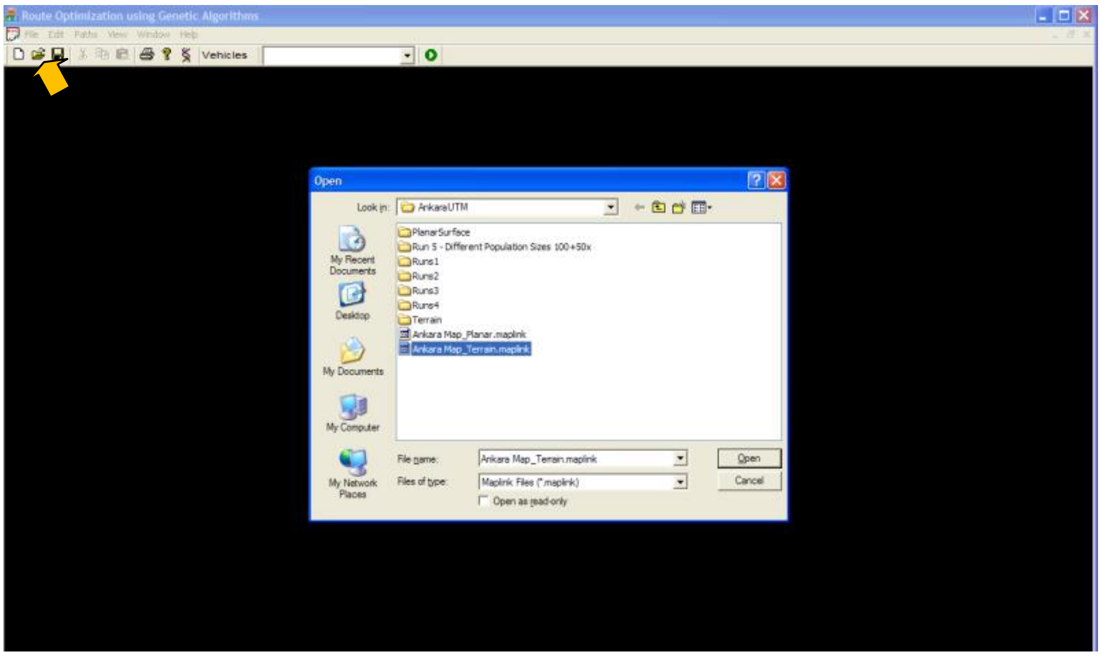
```
C:\Onur\Maps\Shape\AnkaraUTM\PlanarSurface\AllRoadsUTM_polyline.shp
C:\Onur\Maps\Shape\AnkaraUTM\BinLocations_point.shp
```


Ankara Map_Terrain.maplink - Notepad
File Edit Format View Help
Ln 1, Col 1

```
C:\Onur\Maps\Shape\AnkaraUTM\Terrain\AllRoadsUTM_polyline.shp
C:\Onur\Maps\Shape\AnkaraUTM\BinLocations_point.shp
C:\Onur\GGIT\Thesis\DEM\srtm\out.grd
```

Figure A. 1 – Maplink files

Opening the appropriate maplink file, the application loads the shape files and grid file.



Look in: AnkaraUTM

PlanarSurface
Run 5 - Different Population Sizes 100+50x
Runs1
Runs2
Runs3
Runs4
Terrain
Ankara Map_Planar.maplink
Ankara Map_Terrain.maplink

File name: Ankara Map_Terrain.maplink
Files of type: Maplink Files (*.maplink)
☐ Open as read-only

Figure A. 2 – Open Dialog

Once the map opens, the user can pan through the 3D terrain map and see the location of the garbage collection points as well as the landfill.

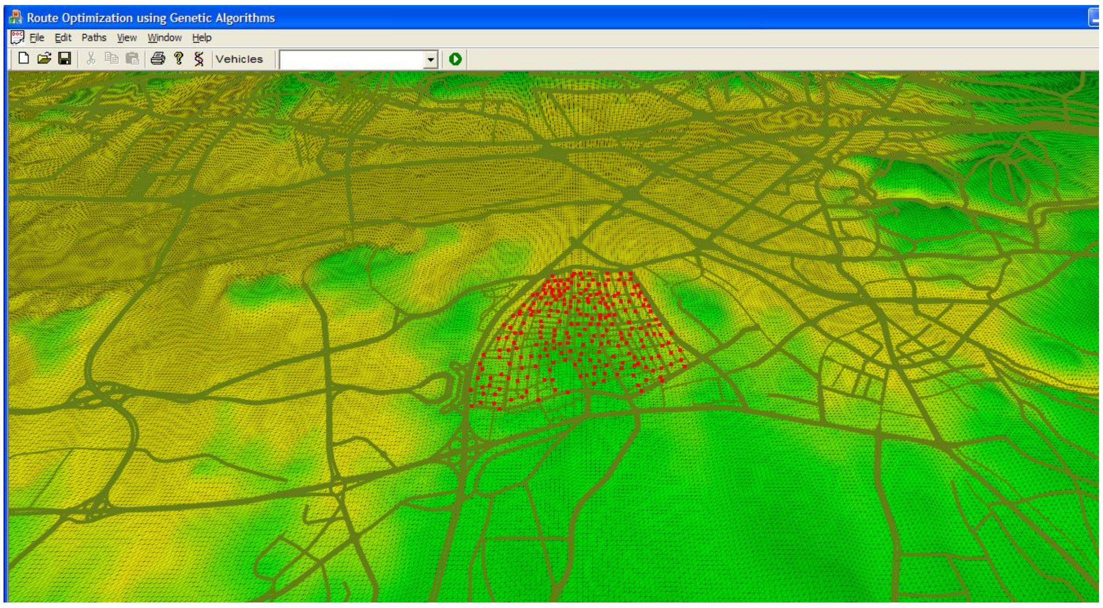
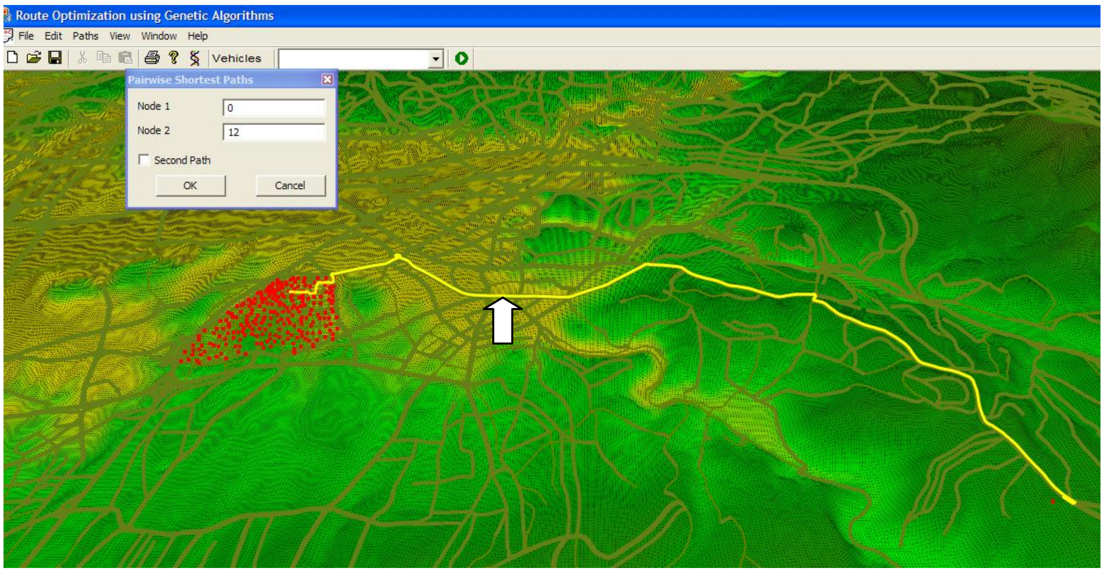

Route Optimization using Genetic Algorithms
File Edit Paths View Window Help
Vehicles

Figure A. 3 – Garbage Collection Points on 3D Terrain Map

If the application runs on the shape files for the first time, the calculation of the pairwise shortest paths along with other pre-processing stages takes a few minutes to complete. Once the shortest paths are computed, they are recorded in the same folder as the shape files to speed-up the process of opening the map for the next time.

The application enables the user to view the shortest paths between any desired nodes individually. The shortest path dialog can be opened by the shortcut Ctrl+S and typing the appropriate node numbers of the source and the destination. The node number 0 denotes the landfill and the remaining 239 nodes denote the points in the same order as they exist in the shape file.



Node 1: 0
Node 2: 12

Figure A. 4 – Displaying Shortest Path

The genetic optimization algorithm can be started by clicking the 'Genetic' toolbar button marked in Figure A. 5.

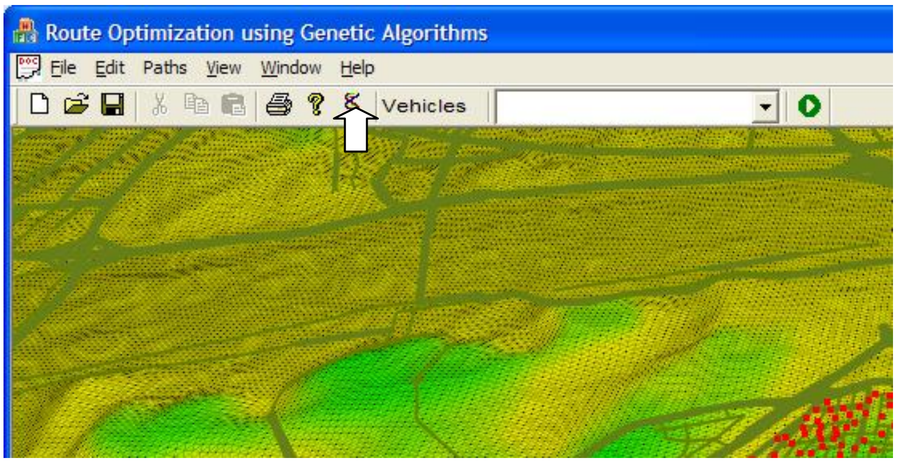
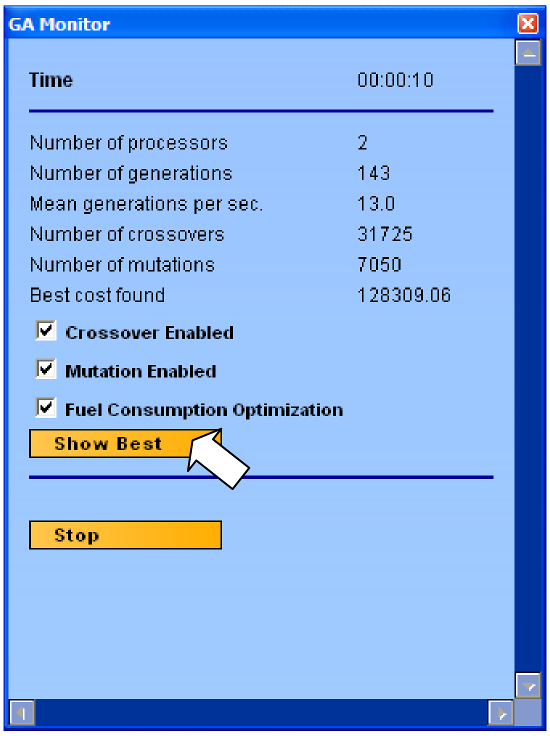


Figure A. 5 – Starting the Genetic Algorithm

As the program is designed as a multi-thread application, the progress of the genetic algorithm can be monitored and visualized at any time. The GA monitor displays the current progress of the genetic algorithm. The operation of the genetic algorithm can also be controlled by means of the check boxes on the GA monitor. For instance, the crossover and mutations can be enabled/disabled any time by the user. Furthermore, the genetic algorithm can be switched between distance optimization mode to fuel consumption optimization mode by using the "Fuel Consumption Optimization" checkbox.

Clicking on the Show Best button on the GA monitor displays the best route found so far (Figure A. 6).


GA Monitor

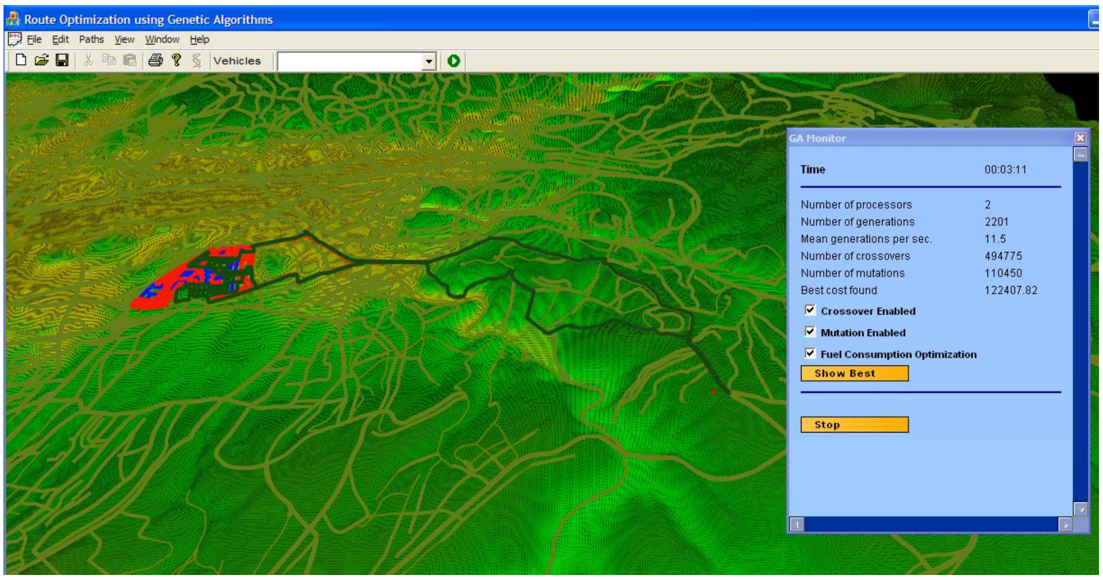
Time	00:00:10
Number of processors	2
Number of generations	143
Mean generations per sec.	13.0
Number of crossovers	31725
Number of mutations	7050
Best cost found	128309.06

☑ Crossover Enabled

☑ Mutation Enabled

☑ Fuel Consumption Optimization

Figure A. 6 – Genetic Algorithm Monitor


Route Optimization using Genetic Algorithms
File Edit Paths View Window Help
Vehicles
GA Monitor
Show Best
Stop

```
Time                              00:03:11

Number of processors              2
Number of generations             2201
Mean generations per sec.         11.5
Number of crossovers              494775
Number of mutations               110450
Best cost found                   122407.82
[✓] Crossover Enabled
[✓] Mutation Enabled
[✓] Fuel Consumption Optimization
```

Figure A. 7 – Visualization of the Best Route

Even though the genetic algorithm has a termination criterion, it can also be terminated by clicking Stop on the GA Monitor Dialog.

Clicking on the play button on the toolbar, it is possible to simulate the movements of the vehicles

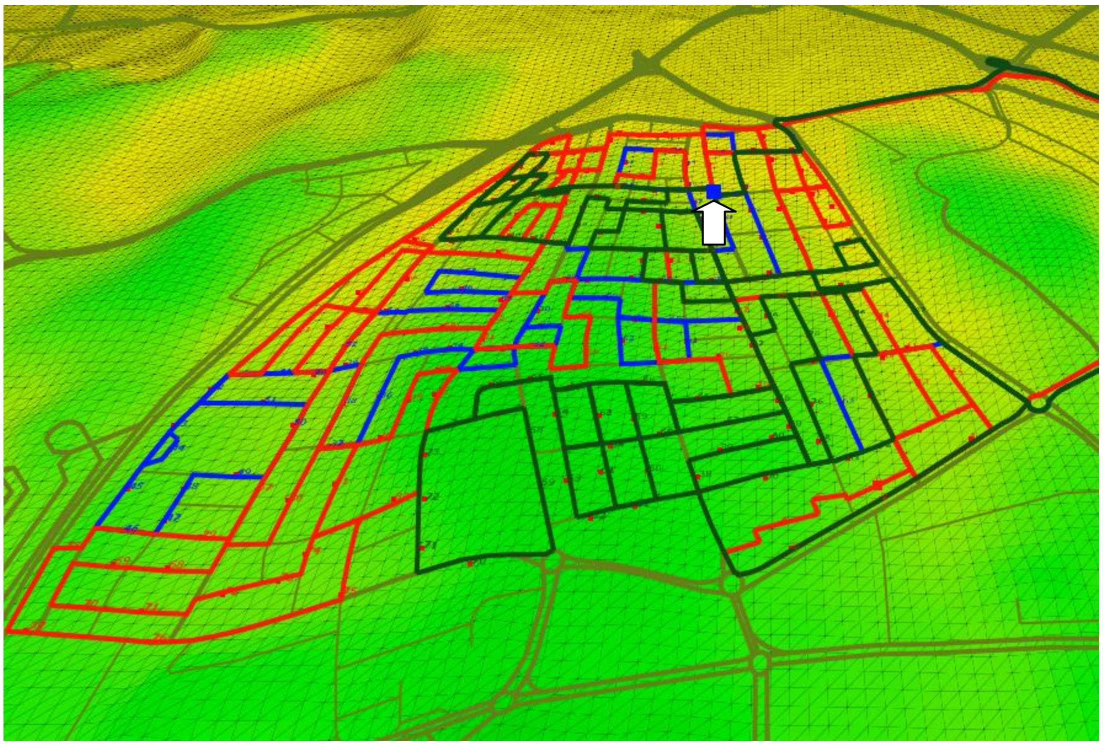


Figure A. 8 – Simulation of the Vehicle on its Path

APPENDIX B

SOURCE CODE FRAGMENTS

Data Structures Used Within the Code

```
struct t_Point {
      double x;
      double y;
      t_Point(double px, double py) : x(px),  y(py) {}
      t_Point() {}
      t_Point operator -(const t_Point& rhs) const {return
t_Point(x - rhs.x,y - rhs.y);}
      double operator *(const t_Point &rhs)
      const {return x*rhs.x+y*rhs.y;}     // dot product
      double magnitude() const {return hypot(x,y);}
};

struct t_Route {
      vector<int> vPoints;
      double dCost;
      double dPenalty;
};

struct t_Chromosome {
      int                           nGlobalThreadID;
      int                           nRank;
      double                        dNumAlternatePaths;
      vector<t_Route>               vRoutes;
      double                        dCost;
      double                        dPenalty;
      double                        dFitness;
      bool operator <(const t_Chromosome &rhs) const
      {return dCost < rhs.dCost;}
};

struct t_GAResources
{
      double      dCrossover;
      double      dMutation;
      double      dElitism;
      int         nPopulationSize;
      vector <int>      vDefaultVehicles;
      vector<t_Point *> vPoints;
      vector<t_Point *> vVehicleLocations;
      vector< vector<t_Path> > vShortestPaths;
      vector< vector<t_Path> > vSecondShortestPaths;
      t_Point                         ptGarage;

};

struct t_Path {
      double dCost;      //Cost of first shortest path
      double dAverageSlope;
      vector<t_Point> vPoints;        // Points of shortest path
      vector<int> vNodesPassed;       // Significant nodes that are
visited between source and destination
      void *pFirstEdge; // The first edge of the shortest path
      t_Path() {dCost=0;pFirstEdge=NULL;}
};
```

Declaration of Genetic Algorithm Functions

```
      void Initialize();
      void CreateRandomChromosome(t_Chromosome *pThisChromosome);

      int RouletteWheelSelection( double *pDistanceMatrix,
         char *pSelectionStatusMatrix = NULL,
         double dExp=-3.0);
      int RouletteWheelSelectionExt( int nCurrentPoint,
         char *pSelectionStatusMatrix,
         double dExp=-3.0,
         bool bCheckPassedNodes=true);

      void CalculateFitness (vector<t_Chromosome *> &vChromosomes);
      void Crossover(t_Chromosome *pParent1,
                     t_Chromosome *pParent2,
                     t_Chromosome *pChild1);
      void OptimizeSequence(
            t_Chromosome *pChromosome /*pointer to chromosome*/,
            int nVehicleNo    /* Required if nStart=0*/,
            int nStart,
            int nEnd,
            int   nNumActiveNeighbours=3 /* the number of nearest
neighbours to check */,
            double      dLimitingDistanceRatio=2.0 /* omit the
neighbour if its distance is larger than NN*ratio */,
            int nTimeLimit=0 /* allowed time limit for
optimization*/);
      void Mutation (t_Chromosome *pChromosome);
      void Mutation_2opt(t_Chromosome *pChromosome);
      void SelectChromosomesBasedOnFitness(
         vector<int> &setSelection,
         vector<t_Chromosome *> &vChromosomes,
         int nNumChromosomes,int nNumSample);
      void CalcDistanceMatrix();
      inline double GetSegmentCost(
             int nVehicleNo,
             vector<int> &vPoints,
             int nStart,
             int nEnd);
      inline double GetRouteCost (int nVehicleNo, vector<int>
&vPoints);

      void Run();
private:
      inline void ValidateCost(t_Chromosome *pChromosome);
      inline void CalculatePenalty(t_Chromosome *pChromosome);
```

Implementation of Mutation Function

```
void CGeneticAlgorithm::Mutation (t_Chromosome *pChromosome)
{

      ValidateCost(pChromosome);
      int i,j;
      int nNodeCount = m_nPointCount+1;
      int *pnVehicleNumbers = new int[m_nPointCount];
      int *pnPointOrder = new int [m_nPointCount];

      static int nPassCount=0;
      nPassCount++;
      ASSERT (pChromosome->dCost>0);



      /*
            Indexing the points-vehicle assignment
      */

      for (i=0;i<m_nVehicleCount;i++)
      {
            t_Route &thisRoute = pChromosome->vRoutes[i];
            for (j=0; j<thisRoute.vPoints.size(); j++)
            {
                  int nPointNo = thisRoute.getPointNoAt(j);
                  if (nPointNo != DEPONI)
                  {
                        pnVehicleNumbers[ nPointNo ] = i;
                        pnPointOrder[nPointNo] = j;
                  }
            }
      }



      struct t_PointNeighbourhood {
            int nVehicleNo;
            int nRouteSize;
            int nPointNo;
            int nPrevPointNo;
            int nNextPointNo;
            int nPointPos;
            double dCost;
      };

      for (i=0; i < m_nPointCount/4; i++)
      {
            t_PointNeighbourhood p1,p2;

            p1.nVehicleNo = ( rand() % m_nVehicleCount);
            p1.nRouteSize = pChromosome-
>vRoutes[p1.nVehicleNo].vPoints.size();
            p1.nPointPos = (rand() % p1.nRouteSize );
            p1.nPointNo = pChromosome-
>vRoutes[p1.nVehicleNo].getPointNoAt( p1.nPointPos );
            p1.nPrevPointNo = (p1.nPointPos==0) ? NOT_EXISTS :
pChromosome->vRoutes[p1.nVehicleNo].getPointNoAt( p1.nPointPos-1 );
            p1.nNextPointNo = (p1.nPointPos == p1.nRouteSize-1) ?
NOT_EXISTS : pChromosome->vRoutes[p1.nVehicleNo].getPointNoAt(
p1.nPointPos+1 );
            if (p1.nPointNo == DEPONI) continue;
            p2.nPointNo = RouletteWheelSelectionExt(p1.nPointNo,
NULL);

            p2.nVehicleNo = pnVehicleNumbers[p2.nPointNo];
            p2.nPointPos = pnPointOrder[p2.nPointNo];
            p2.nRouteSize = pChromosome-
>vRoutes[p2.nVehicleNo].vPoints.size();
            p2.nPrevPointNo = (p2.nPointPos==0) ? NOT_EXISTS :
pChromosome->vRoutes[p2.nVehicleNo].getPointNoAt( p2.nPointPos-1 );
            p2.nNextPointNo = (p2.nPointPos == p2.nRouteSize-1) ?
NOT_EXISTS : pChromosome->vRoutes[p2.nVehicleNo].getPointNoAt(
p2.nPointPos+1 );

            if (p1.nRouteSize<=2 || p2.nRouteSize<=2) continue;



      //    if (p1.nRouteSize)
            if (p1.nVehicleNo == p2.nVehicleNo && abs
(p1.nPointPos-p2.nPointPos)<6)
            {     // If the points to try are in the same route and
adjacent

            }     // If the points to try are in the same route and
adjacent
            else
            {     // The points are not on the same route OR are
not adjacent
                  int pnCurrentRouteV1[6];
                  int pnCurrentRouteV2[6];
                  int pnMutationRouteV1[6];
                  int pnMutationRouteV2[6];

                  int nV1StartPos = p1.nPointPos-2;
                  int nV1EndPos     = p1.nPointPos+3;
                  int nV2StartPos = p2.nPointPos-2;
                  int nV2EndPos     = p2.nPointPos+3;
                  if (nV1StartPos == -2) nV1StartPos = NOT_EXISTS;
                  if (nV2StartPos == -2) nV2StartPos = NOT_EXISTS;
                  if (nV1EndPos > p1.nRouteSize-1) nV1EndPos =
p1.nRouteSize-1;
                  if (nV2EndPos > p2.nRouteSize-1) nV2EndPos =
p2.nRouteSize-1;
```

```
                  int nV1Count = nV1EndPos-nV1StartPos+1;
                  int nV2Count = nV2EndPos-nV2StartPos+1;

                  t_Route &firstRoute = pChromosome-
>vRoutes[p1.nVehicleNo];
                  t_Route &secondRoute = pChromosome-
>vRoutes[p2.nVehicleNo];
                  int nIndex = 0;   // index of partial route array

                  for (j=nV1StartPos, nIndex=0; j<=nV1EndPos; j++,
nIndex++)
                  {
                        if (j==-1 || j== p1.nRouteSize-1)
pnCurrentRouteV1[nIndex] = NOT_EXISTS;    // Starting from Garage
                        else pnCurrentRouteV1[nIndex] =
firstRoute.vPoints[j];
                        if (j== p1.nPointPos)
pnMutationRouteV1[nIndex] = secondRoute.vPoints[p2.nPointPos];
      // if point of mutation
                        else pnMutationRouteV1[nIndex] =
pnCurrentRouteV1[nIndex];     // if not point of mutation, same as
current route
                  }

                  for (j=nV2StartPos, nIndex=0; j<=nV2EndPos; j++,
nIndex++)
                  {
                        if (j==-1) pnCurrentRouteV2[nIndex] =
NOT_EXISTS; // Starting from Garage
                        else pnCurrentRouteV2[nIndex] =
secondRoute.vPoints[j];
                        if (j== p2.nPointPos)
pnMutationRouteV2[nIndex] = firstRoute.vPoints[p1.nPointPos];
      // if point of mutation
                        else pnMutationRouteV2[nIndex] =
pnCurrentRouteV2[nIndex];     // if not point of mutation, same as
current route

                  }
                  static int nCycleCount=0;
                  nCycleCount++;

                  AdjustPathDirections(pnMutationRouteV1,nV1Count);
                  AdjustPathDirections(pnMutationRouteV2,nV2Count);

                  // The last edges of the mutated and original
partial routes must be the same
                  // so that the path can continue without a U-Turn
                  // Otherwise, skip the mutation
                  int nV1Last = nV1Count-1;     // index of the
last point of mutation zone
                  int nV2Last = nV2Count-1;
                  if (pnMutationRouteV1[nV1Last]==DEPONI) {nV1Last-
-; nV1Count--;}
                  if (pnMutationRouteV2[nV2Last]==DEPONI) {nV2Last-
-; nV2Count--;}

                  if (pnMutationRouteV1[nV1Last-1] !=
pnCurrentRouteV1[nV1Last-1]) continue;
```

```
                  if (pnMutationRouteV1[nV1Last] !=
pnCurrentRouteV1[nV1Last]) continue;
                  if (pnMutationRouteV2[nV2Last-1] !=
pnCurrentRouteV2[nV2Last-1]) continue;
                  if (pnMutationRouteV2[nV2Last] !=
pnCurrentRouteV2[nV2Last]) continue;


                  double dCostV1BeforeMutation =
GetPartialCost(pnCurrentRouteV1, nV1Count);
                  double dCostV2BeforeMutation =
GetPartialCost(pnCurrentRouteV2, nV2Count);
                  double dCostV1AfterMutation =
GetPartialCost(pnMutationRouteV1, nV1Count);
                  double dCostV2AfterMutation =
GetPartialCost(pnMutationRouteV2, nV2Count);

                  double dCostRatio =
(dCostV1BeforeMutation+dCostV2BeforeMutation) /
(dCostV1AfterMutation+dCostV2AfterMutation);
                  bool bApplyMutation = dCostRatio>1.0;
                  if (!bApplyMutation)
                  {
                        if (dCostRatio>0.95)
                              if (pChromosome->nRank>50)
bApplyMutation=(RAND01>0.5);
                  }

                  if (dCostV1AfterMutation + dCostV2AfterMutation <
dCostV1BeforeMutation + dCostV2BeforeMutation)
                  {     // Mutation is beneficial
                        for (j=nV1StartPos, nIndex=0; j< nV1EndPos;
j++, nIndex++)
                        {
                              if (j<0) continue;

      firstRoute.vPoints[j]=pnMutationRouteV1[nIndex];
                        }
                        for (j=nV2StartPos, nIndex=0; j< nV2EndPos;
j++, nIndex++)
                        {
                              if (j<0) continue;

      secondRoute.vPoints[j]=pnMutationRouteV2[nIndex];
                        }
                        double dDeltaCostV1 = dCostV1AfterMutation
- dCostV1BeforeMutation;
                        double dDeltaCostV2 = dCostV2AfterMutation
- dCostV2BeforeMutation;
                        firstRoute.dCost += dDeltaCostV1;
                        secondRoute.dCost += dDeltaCostV2;
                        pChromosome->dCost += dDeltaCostV1 +
dDeltaCostV2;

                        pnVehicleNumbers[p1.nPointNo] =
p2.nVehicleNo;
                        pnVehicleNumbers[p2.nPointNo] =
p1.nVehicleNo;
                        pnPointOrder[p1.nPointNo] = p2.nPointPos;
                        pnPointOrder[p2.nPointNo] = p1.nPointPos;
```

```
                         ValidateCost(pChromosome);

                   }     // if mutation is beneficial

             }     //else



       }     // for i

       delete [] pnVehicleNumbers;
       delete [] pnPointOrder;

       ValidateCost(pChromosome);
}
```

Implementation of Crossover Function

```
void CGeneticAlgorithm::Crossover(t_Chromosome
*pParent1,t_Chromosome *pParent2,t_Chromosome *pChild1)
{
       ValidateCost(pParent1);
       ValidateCost(pParent2);

       static long nCrossoverCount = 0;
       nCrossoverCount++;
       // Edge Recombination Crossover
       int i,j,k;

       int nNodeCount = m_nPointCount + 1;
       t_Vector *pPathDirections = new t_Vector[m_nVehicleCount];

       struct t_Neighbours {
             int nCount;
             int arNeighbours[8];
             t_Neighbours() {nCount=0;}
       };
       t_Neighbours *pEdges = new t_Neighbours[m_nPointCount];



       for (i=0; i<m_nVehicleCount; i++)
       {
             t_Route *thisRoute = &pParent1->vRoutes[i];
             int nRoutePointCount=thisRoute->vPoints.size();
             for (j=0;j<nRoutePointCount-1;j++)
             {     // Edges of first parent
                   int n1 = thisRoute->getPointNoAt(j);
                   int n2 = thisRoute->getPointNoAt(j+1);
                   if (n1!=DEPONI && n2!=DEPONI)
                   {
                         ASSERT (pEdges[n1].nCount<=8);
                         pEdges[n1].arNeighbours[ pEdges[n1].nCount++
 ] = n2;
                         pEdges[n2].arNeighbours[ pEdges[n2].nCount++
 ] = n1;
                   }
             }
             thisRoute = &pParent2->vRoutes[i];
             nRoutePointCount=thisRoute->vPoints.size();
             for (j=0;j<nRoutePointCount-1;j++)
             {     // Edges of second parent
                   int n1 = thisRoute->getPointNoAt(j);
                   int n2 = thisRoute->getPointNoAt(j+1);

                   if (n1!=DEPONI && n2!=DEPONI)
                   {
                         ASSERT (pEdges[n1].nCount<=8);
                         pEdges[n1].arNeighbours[ pEdges[n1].nCount++
 ] = n2;
                         pEdges[n2].arNeighbours[ pEdges[n2].nCount++
 ] = n1;
                   }
             }
       }

       int nVehicleNo;
       long
 nEstimatedPointPerVehicle=(m_nPointCount*1.20)/m_nVehicleCount;
       char *pSelectionStatus=new char[m_nPointCount];
       int *pnSelectedPoints=new int[m_nVehicleCount]; // keeps track
 of last selected point of each vehicle route

       pChild1->dCost=0;
       pChild1->dPenalty=0;
       for (i=0; i<m_nVehicleCount; i++)
       {
             pChild1->vRoutes[i].dCost = 0;      // initializing cost
             pChild1->vRoutes[i].vPoints.reserve(
 nEstimatedPointPerVehicle ); // reserving space to avoid frequent
 memory allocations
             pnSelectedPoints[i] = NOT_EXISTS;   // no point selected
 yet for the vehicle
       }

       for (i=0; i<m_nPointCount; i++) pSelectionStatus[i] =
 POINT_NOT_SELECTED;     // marking as not selected

       int nSelectedPoint;     // keeps track of the current selected
 point
       for (i=0;i<m_nPointCount;i++)
       {
             nVehicleNo=0;
             double dLowestCost = 1e99;
             for (j=0; j < m_nVehicleCount; j++)
             {
                   t_Route &thisRoute = pChild1->vRoutes[j];
                   if (thisRoute.dCost < dLowestCost)
                   {
                         nVehicleNo=j;     // Try to find the vehicle
 which traveled less
                         dLowestCost=thisRoute.dCost;
                   }
             }
             t_Route &childRoute = pChild1->vRoutes[nVehicleNo];
             if (pnSelectedPoints[nVehicleNo] == NOT_EXISTS)
             {      // No point has been assigned to this vehicle's
 route yet,
                   // starting from the position of the vehicle
                   if ( (rand() % 2 ) == 0) nSelectedPoint =
 pParent1->vRoutes[nVehicleNo].getPointNoAt(0);
                   else nSelectedPoint = pParent2-
 >vRoutes[nVehicleNo].getPointNoAt(0);
                   if ( pSelectionStatus[nSelectedPoint] == 1) // if
 no available neighbour point from parents, select another

       nSelectedPoint=RouletteWheelSelectionExt(GARAGE,pSelectionStat
 us);

                   pSelectionStatus[nSelectedPoint] = POINT_SELECTED;
       // mark the point as selected to prevent re-selection
                   pnSelectedPoints[nVehicleNo]=nSelectedPoint;
                   childRoute.vPoints.push_back( nSelectedPoint );
                   childRoute.dCost =
 GetCostVP(nVehicleNo,nSelectedPoint);
                   pPathDirections[nVehicleNo] =
 m_pPathEndVectors[Point2Node(nSelectedPoint)];
             }
             else
             {      // starting from the last point visited
                   nSelectedPoint=pnSelectedPoints[nVehicleNo];    //
 last selected point for the vehicle

                   t_Neighbours *pNeighbours =
 &pEdges[nSelectedPoint];
                   int         nNumNeighbours = pNeighbours->nCount;
                   int         nSelectedNeighbour = NOT_EXISTS;
                   double      dSelectedNeighbourDist = 1e99;
                   for (j=0; j<nNumNeighbours; j++)
                   {
                         int nThisPoint = pNeighbours-
 >arNeighbours[j];
                         if (pSelectionStatus[nThisPoint] ==
 POINT_NOT_SELECTED)
                         {
                               double dDistance =
 GetCostPP(nSelectedPoint,nThisPoint);
                               if (dDistance <
 dSelectedNeighbourDist)
                               {
                                     nSelectedNeighbour = nThisPoint;
                                     dSelectedNeighbourDist =
 dDistance;
                               }
                         }
                   }
                   if (nSelectedNeighbour == NOT_EXISTS)
                   {

       nSelectedNeighbour=RouletteWheelSelectionExt(nSelectedPoint,pS
 electionStatus);
                   }
                   t_Vector &v1 = pPathDirections[nVehicleNo];
                   t_Vector &v2 =
 m_pPathStartVectors[Point2Node(nSelectedPoint)*nNodeCount+Point2Node
 (nSelectedNeighbour)];
                   double dCosTheta=(v1*v2) / (v1.magnitude() *
 v2.magnitude() );

                   if (dCosTheta < -.96)
                   {
                         childRoute.vPoints.back() |= 0x10000;
                         pPathDirections[nVehicleNo] =
 m_pSecondPathEndVectors[Point2Node(nSelectedPoint)*nNodeCount+Point2
 Node(nSelectedNeighbour)];
                   }
                   else
                   {
                         pPathDirections[nVehicleNo] =
 m_pPathEndVectors[Point2Node(nSelectedPoint)*nNodeCount+Point2Node(n
 SelectedNeighbour)];
                   }
                   pSelectionStatus[ nSelectedNeighbour ] =
 POINT_SELECTED;
                   double dCost =
 GetCostPP(childRoute.vPoints.back(),nSelectedNeighbour);
                   ASSERT(dCost>0);
                   childRoute.dCost+= dCost;
                   childRoute.vPoints.push_back( nSelectedNeighbour
 );
                   pnSelectedPoints[ nVehicleNo ]=nSelectedNeighbour;
             } //  (pnSelectedPoints[nVehicleNo] != NOT_EXISTS)

       }     //for (i=0;i<m_nPointCount;i++)

       for (i=0;i<m_nVehicleCount;i++)
       {
             t_Route &thisRoute = pChild1->vRoutes[i];
             int nLastPoint = thisRoute.vPoints.back() & 0xFFFF;
             t_Vector &v1 = pPathDirections[i];
             t_Vector &v2 =
 m_pPathStartVectors[Point2Node(nLastPoint)*nNodeCount+0];

             double dCosTheta=(v1*v2) / (v1.magnitude() *
 v2.magnitude() );

             if (dCosTheta < -.96)
             {
                   thisRoute.vPoints.back() |= 0x10000;
             }
             double dCost = GetCostPG(thisRoute.vPoints.back() );
             ASSERT(dCost>0);
             thisRoute.dCost += dCost;
             pChild1->dCost += thisRoute.dCost;
       }

       delete [] pPathDirections;
       delete [] pnSelectedPoints;
       delete [] pSelectionStatus;
       delete [] pEdges;

       ValidateCost(pParent1);
       ValidateCost(pParent2);
       ValidateCost(pChild1);

 } // Crossover
```

Implementation of CalculateFitness Function

```
      void CGeneticAlgorithm::CalculateFitness
            (vector<t_Chromosome *> &vChromosomes)
{
      int nPopulationCount=vChromosomes.size();
      double dSum=0;

      double dMaxCost =
      vChromosomes.back()->dCost +  vChromosomes.back()->dPenalty;

      for (int i=0;i<nPopulationCount;i++)
      {
            double dCost=vChromosomes[i]->dCost+
                         vChromosomes[i]->dPenalty;
            vChromosomes[i]->dFitness =
            (nPopulationCount-i) / (dCost/dMaxCost);
            dSum += vChromosomes[i]->dFitness;
      }
      for (int i=0; i<nPopulationCount; i++)
      {
            vChromosomes[i]->dFitness /= dSum;
      }


}
```

Implementation of Mutation 2-opt Function

```
      void CGeneticAlgorithm::Mutation_2opt
                   (t_Chromosome      *pChromosome)
{
      int nVehcileNo;
      int i,j,k;
      int nNumVehcles = pChromosome->vRoutes.size();
      DWORD d=GetTickCount();
      for (nVehcileNo=0; nVehcileNo < nNumVehcles; nVehcileNo++)
      {
            t_Route &thisRoute = pChromosome->vRoutes[nVehcileNo];
            int nNumEdges = thisRoute.vPoints.size()-1;
            double dAverageEdgeSize = thisRoute.dCost / nNumEdges;

            int *pOriginal = new int[thisRoute.vPoints.size()];
            int *pModified = new int[thisRoute.vPoints.size()];
            double dRouteCost = thisRoute.dCost;
            for (int *p1=pOriginal,  *p2=pModified,k=0;
k<=nNumEdges; k++,p1++,p2++ ) *p1=*p2=thisRoute.vPoints[k];

            vector<int>::iterator
                    it_route = thisRoute.vPoints.begin();
            for (i=1; i<nNumEdges-1; i++)
            {
                  for (j=i+1; j<nNumEdges;j++)
                  {
                        double dDistPoints =
GetCostPP(pOriginal[i], pOriginal[j]);
                        if (dDistPoints>dAverageEdgeSize) continue;
                        if (j-i==1)
                        {
                              SWAP(pModified[i],pModified[j]);
                        }
                        else
                        {

                              for (k=i+1; k<j;k++)
                              {

                                    ASSERT(pModified[k]!=GARAGE);
      // requires change
                                    pModified[k]=pOriginal[j-k+i];
      // Inverting part of sequence
                              }


                        }


                        int nStart = i-1;
                        int nEnd = j+1;

                        double dCostAfterRepair =
RepairRoute(pModified,pOriginal,nNumEdges+1);

                        if (dCostAfterRepair<dRouteCost)
                        {
                              for (k=nStart; k<=nEnd;k++)
pOriginal[k]=pModified[k];
                              dRouteCost = dCostAfterRepair;
                        }
                        else
                        {
                              for (k=nStart; k<=nEnd;k++)
pModified[k]=pOriginal[k];
                        }

                  } // j
            } // i
            stdext::hash_map<int,int> hStatus;
            vector<int> vModified(nNumEdges+1);
            for (k=0;k<=nNumEdges;k++)
            {
                  vModified[k]=pModified[k];
                  ASSERT(hStatus[ pModified[k] ] != 1);
                  hStatus[ pModified[k] ] = 1;
```

```
			}

			if (dRouteCost<thisRoute.dCost)
			{
				for (k=0; k<=nNumEdges;k++)
					thisRoute.vPoints[k]=pModified[k];
				double dCost =
GetRouteCost(nVehcileNo,thisRoute.vPoints);
				pChromosome->dCost += (dCost-thisRoute.dCost);
				thisRoute.dCost = dCost;
			}
			delete [] pOriginal;
			delete [] pModified;
		}	// for nVehicleNo
		DWORD nDiff = GetTickCount() -d;

}
```