

AN EMPLOYEE TRANSPORTING PROBLEM AND ITS HEURISTIC SOLUTIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

İLTER ÖNDER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

DECEMBER 2007

Title of the Thesis : **An Employee Transporting Problem and Its Heuristic Solutions**

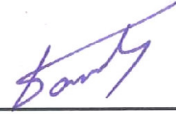
Submitted by **İlter Önder**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University



Prof. Dr. Özhan Uluatam
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Levent Kandiller
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

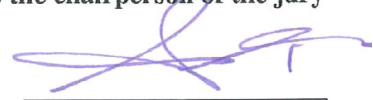


Supervisor
Prof. Dr. Ümit Yüceer

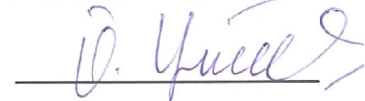
Examination Date : 06.12.2007

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Assis. Prof. Dr. Ferda CanÇetinkaya (Çankaya Univ.)



Prof. Dr. Ümit Yüceer (Çankaya Univ.)



Assis. Prof. Dr. Suat Kasap (Hacettepe Univ.)



STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : İlter Önder

Signature :



Date : 6/12/2007

ABSTRACT

AN EMPLOYEE TRANSPORTING PROBLEM AND ITS HEURISTIC SOLUTIONS

Önder, İlter

M.Sc., Department of Industrial Engineering

Supervisor: Prof. Dr. Ümit Yüceer

December 2007, 154 pages

A typical Vehicle Routing Problem (VRP) can be described as a problem of designing the least cost routes from one depot to a set of geographically scattered points. The VRP assumes that the vehicle capacities are identical, yet in real life the vehicle capacities are different. Therefore, this thesis presents a study of VRP with heterogeneous vehicles (HVRP). A lower bound on the cost of vehicles and routing is calculated for the HVRP using some mathematical models. Various heuristics are attempted to decide which one provides better solutions on the average. The better heuristic is selected based on the deviation from the lower bound. A simple software is prepared using the best heuristic methods for the employee pickup and delivery operations of a hypothetical company.

Keywords: Heterogeneous Vehicle Routing Problem, Heuristic Methods

ÖZ

BİR ÇALIŞAN TAŞINMASI PROBLEMİ VE SEZGİSEL ÇÖZÜMLERİ

Önder, İlter

Yüksek lisans, Endüstri Mühendisliği Anabilim Dalı

Tez Yöneticisi : Prof. Dr. Ümit Yüceer

Aralık 2007, 154 sayfa

Tipik bir taşıt güzergâhı rotalama (TGR) coğrafi olarak birbirinden ayrı noktalara en az maliyetle ulaşan rotaların bulunmasına yöneliktir. TGR araç kapasitelerini eşit kabul eder, ancak Gerçek hayatta araç kapasiteleri genellikle farklıdır. Bu tezde birbirinden farklı kapasiteli (türdeş olmayan) araçların rotalanması üzerine bir çalışma sunulmuştur. Türdeş olmayan taşıt güzergâhı problemi için matematik modelleme kullanılarak bir alt sınır belirlenmiştir. Daha sonra sezgisel yöntemler kullanılarak çözüm yöntemleri geliştirilmiş, alt sınıra yakınlıklarına göre sezgisel yöntemler arasında bir seçim yapılmıştır. Teorik bir firmanın çalışan dağıtımı ve toplanması için en iyi sonucu veren sezgisel yöntemleri içeren basit bir yazılım hazırlanmıştır.

Anahtar Kelimeler: Türdeş olmayan Araç Rotalama Problemi, Sezgisel Yöntemler

ACKNOWLEDGEMENTS

Firstly I would like to express my gratitude to Prof. Dr. Ümit Yüceer for his support and understanding. I would like to thank my instructors Özge Uncu and Faruk Polat for widening my point of view. I would like to thank my jury members Asist. Prof. Dr. Ferda Can Çetinkaya and Asist. Prof. Dr. Suat Kasap for their valuable comments and patience. This study would not be possible without Prof. Dr. Fetih Yıldırım.

I would like to thank my chairman Prof. Dr. Levent Kandiller, for his tolerance and support for my graduate studies. Moreover, I would like to thank him as the dean of the faculty, as this work would not be complete without the PCs of CAD/CAM LAB of Çankaya University Industrial Engineering Department.

I dedicate this work to all members of my loving family and caring extended family. This page is not enough to express my thanks to them. I present my special thanks to my sister Emel Önder, for making my life easier, for her love, understanding and support and care. I would like to thank my brother İsmail Özkan for his support and encouraging curiosity.

I would like to thank my colleagues Engin Topan, İpek Seyran Topan, and Miray Hanım Aslan for their support, valuable help and encouragement. I would like to thank my colleagues Ender Yıldırım, Dilan Karatepe and Serdar Soysal for their support and help.

I would like to thank all my friends, especially Koray Kadiođlu and İrfan Karaca for their help. Special thanks to Aytunç Göy and Aslı Erdođ, for their existence and help and company.

Last, but not least, I would like to thank my friend and student Kubilay Volkan Kaygısız, for his help, support and care, when I was really in need.

Thanks to everyone whoever, I have shared my time, and who have helped me become who I am now.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM.....	iii
ABSTRACT.....	iv
ÖZ.....	v
ACKNOWLEDGEMENTS.....	vi
TABLE OF CONTENTS.....	vii
TABLE OF TABLES.....	ix
TABLE OF FIGURES.....	x
CHAPTERS	
1.INTRODUCTION.....	1
2.REVIEW OF VRP SOLUTION TECHNIQUES AND EXTENTIONS TO HVRP.....	5
2.1.Solution Methods for the VRP.....	6
2.1.1.Exact Solution Methods for VRP.....	6
2.1.2.Heuristic Solution Methods for VRP.....	7
2.1.3.Metaheuristics for VRP.....	9
2.2.An Extention to HVRP and Solution Methods for HVRP.....	13
3. A MATHEMATICAL MODEL FOR EMPLOYEE TRASPORTING PROBLEM.....	19
3.1.Statement and Formulation of the Vehicle Routing Problem.....	19
3.2.Assumptions of the Routing Problem.....	20
3.3.Mathematical Model of the Routing Problem.....	21
3.4.Initial Experiments.....	24
3.5.Results of the Initial Experiments.....	26
3.6.Linear Relaxation of the Mathematical Model.....	26
4.IMPLEMENTED ALGORITHMS.....	30
4.1.Route-first-cluster-second Algorithm for HVRP.....	30

4.2.Cluster-first-route-second Algorithm for HVRP.....	33
4.3.Genetic Algorithm for HVRP.....	35
4.4.Multi-Agent Solution for HVRP.....	40
5.NUMERICAL EXPERIMENTS AND COMPARISON OF THE RESULTS OF IMPLEMENTED ALGORITHMS.....	46
5.1.Cluster-first-route-second Algoritihm.....	46
5.2.Genetic Algorithm.....	51
5.3.Multi-Agent Solution for HVRP.....	54
5.4.Analysis of the Results of the Experimental Runs.....	56
6.AN APPLICATION OF THE PROPOSED ALGORITHMS.....	58
6.1.Results on a Random Problem.....	59
7.CONCLUSION.....	62
REFERENCES.....	R1
APPENDICY	
A. ANOVA Results for Experimental Runs.....	A1
B. GAMS Models of LP Relaxationions of the Proposed Model.....	A10
C. GAMS Models the Proposed Model.....	A13
D. CFRS C code for Ochi Method	A15
E. C code of the Generic Algorithm.....	A22
F. C code of the Multi Agent Algorithm.....	A52

TABLE OF TABLES

Table 5. 1 Average percent deviation from the LB for sweep algorithm on our model.....	51
Table 5. 2 Minimum percentage deviations from the best-known solutions of problem set 1	52
Table 5. 3 Results of Genetic Algorithm on problem set 1.....	53
Table 5. 4 Results of Genetic Algorithm on problem set 2.....	54
Table 5. 5 Average improvement of deviation from the LB with GA.....	54
Table 5. 6 Percentage deviations of different η values for MA, using our model	55
Table 5. 7 Percentage deviations of different η values for MA, on HVRP instances	56
Table 6. 1 Results of the CFRS algorithm for fort he hypothetical firm	60

TABLE OF FIGURES

Figure 4. 1 Pseudo code for sweep algorithm for HVRP	36
Figure 4. 2 Pseudo code for GA algorithm for HVRP.....	37
Figure 4. 3 A sample union graph for NNX.....	39
Figure 5. 1 Main effect plots for the deviation from the LB for our model.....	48
Figure 5. 2 Interaction plots for the deviation from the LB for our model.....	49
Figure 5. 3 Main effect plots including “cluster” for the deviation from the LB for our model	50
Figure 5. 4 Main effect plots including “cluster” for the deviation from the LB for HVRP	50
Figure 6. 1 The plot of the bus-stops and production plant ofthe hypothetical firm	59
Figure 6. 1 The plot of the resulting clusters, then Random method is used with R=1.....	602

CHAPTER 1

INTRODUCTION

Transportation and delivery of the goods that we consume in our daily lives consists of a costly and time-consuming function of any firm. Avoiding the unnecessary delays and costs during transportation and delivery (including storage) is an important factor for any firm to retain and/or improve competitiveness. Consequently, an application of optimization techniques to transportation and delivery of goods (whenever feasible) is a means of providing a reliable and prompt service at a reasonable cost to customers.

A well known Turkish biscuits producing company was initiated in a Kaizen^{*} study on reduction of the losses faced due to inefficiency problems. One of the problems that a Kaizen group deals is to improve the cost and time of employee delivery and pickup operations. It was noticed that the routes of the buses that brought the employees to the plant were not organized. The firm pays a subcontractor on the basis of the number of seats in each bus, instead of the route length. The manager of the subcontracting firm decides on the routes and the number of people to be carried by each bus. As the firm paid on the seat-basis there are cases when the buses traveled with half capacity, or large buses traveled where small ones could have served, yet the firm had to pay for the whole bus regardless the utilization. The problems were first noticed by the firm's industrial engineers. They noticed that the routes rapidly changed quite often and the employees did not know which bus to take. The Kaizen group could not figure out an analytical solution, but rather

* A Japanese word meaning improvement used as a term in Total Quality Management.

intuitive and simple solutions were attempted with no success. Therefore, this thesis makes an attempt to develop a model and solution approaches to solve this problem.

This firm's problem can be classified as a Vehicle Routing Problem (VRP), where the depot is the factory and the customers are viewed as the bus stops within the city. The aim is to create minimal cost routes for the vehicles, such that all the customers are visited once and the vehicles return to the main depot with a high percentage of fill rate. The demands in the bus stops (number of employees) have some variability and this shall be incorporated in the model procedure. Thus, a lengthy one-time solution is not an adequate solution to the problem, a solution that takes a reasonable amount of time and good enough solution is preferable. The VRP is an NP-Hard problem (Lenstra and Rinnooy Kan, 1981), and even a moderate size problem takes a great deal of time to be solved to optimality. A method that can solve the problem in relatively short time is required, so that the employees can be informed prior to their deliveries or return trips.

A later analysis prevails that the problem, in fact, is more complicated since the buses are not identical and the manager of the subcontractor decides on the type of the vehicles, the clusters of the bus stops and the routes within the city. The problem then resembles the Heterogeneous Vehicle Routing Problem (HVRP) in the literature where there is more than one type of vehicle with different capacities. Each vehicle has a fixed cost, a maximum capacity, and a variable cost. The aim of the HVRP is to find a point that balances the fixed cost and the variable (routing) cost of the vehicles by minimizing the total cost of serving all customers (bus-stops). The problem is formulated for designing the optimum crew in logistics or distribution fleets, thus is also referred as a fleet mix VRP (Salhi and Rand, 1993).

The mathematical model we have developed is also used to calculate lower bounds for problems that cannot be solved due to size restrictions, by loosening the integrality constraints. The model for developed for the HVRP version is also improved using some valid inequalities to tighten the lower bound.

Various solution approaches will be attempted based on HVRP in the thesis. The model we have developed can be used to serve as a guideline for further subcontracting agreements, finding the best fleet mix for the company's problem. The HVRP did not attract the attention of the researchers as much as VRP. Taillard

(1996) states that this is due to the hardness of the problem to be solved compared to the VRP.

Dantzig and Ramser (1959) formulated a truck-dispatching problem and their problem has evolved into the VRP. There are exact solution methods based on mathematical formulations and branching algorithms for VRP. There are heuristic methods that can solve these problems in relatively shorter time. The constraints employ a decomposition/partitioning approach to the problem, separating the routing and clustering operations. Clarke and Wright (1964) method was the first heuristic that handled the problem as a whole in an easy applicable manner. The heuristics are grouped in two wide categories: the cluster-first-route-second heuristics and route-first-cluster-second heuristics. In the first strategy, the customers are clustered using a clustering algorithm, and then each clusters is routed solving a TSP problem, in the second strategy starts with a giant TSP tour visiting all the customers then this route is partitioned into the clusters. The first approach has been reported giving promising results for the VRP.

Metaheuristics are also applied to VRP as natural optimization/improvement mechanisms. The simulated annealing, deterministic annealing, tabu search, genetic algorithms, and neural networks have been successfully applied to the VRP problems. Tabu search of Taillard performed better compared to all heuristics and metaheuristics (Toth and Vigo, 2001a).

The HVRP is more limited in terms of solution attempts. The only exact solution approach was presented by Yaman (2006). Osman and Salhi (1994), Gendreau et al. (1999) and Taillard (1999) present three studies that used a tabu search to solve the problem, while Ochi et al. (1998a) and Lima et al. (2004) used genetic algorithms instead of HVRP specific heuristics.

We implement more than one general solution approach with improvement mechanism to HVRP and aim to asses the ability of simpler approaches on the solution of the problem. Route-first-cluster-second, cluster-first-route-second approaches are implemented on HVRP. A genetic algorithm is also implemented on the HVRP problem. Lastly, multi-agent systems that utilize simple VRP heuristics will be used to solve the HVRP. The results of the heuristics and metaheuristics will

be analyzed and the best heuristic/metaheuristic will be selected for implementation on the problem of the biscuit producing firm.

The problem in the biscuit producing firm is rather large having over 90 bus stops, and the firm uses two different vehicle types, thus the size of the problem is too large to be solved optimally. The efficiency of the final solution to be implemented can be assessed using this lower bound. The lower bound is calculated relaxing the integrality constraints on the mathematical model.

This thesis is organized as follows; Chapter 2 includes the major work done on HVRP. A brief explanation of the VRP solution methods are also included in Chapter 2, as the method used in Chapter 5 are based on efficient VRP solutions. Chapter 3 includes the mathematical model we have used and the results of the lower bound calculations for two sets of problem instances. Chapter 4 includes detailed algorithms of the various heuristics we have implemented. Chapter 5 gives the computational settings and the results of the algorithms compared to the best-known solutions and the lower bounds. Chapter 6 includes the results for the biscuit producing firm, the solution and comparison in terms of deviation from the lower bound. Chapter 7 includes the concluding remarks and the results of this study along with further research directions.

CHAPTER 2

REVIEW OF VRP SOLUTION TECHNIQUES AND EXTENTIONS TO HVRP

Vehicle Routing Problem (VRP) and some well-known solution methods in the literature for the VRP are described at the beginning of this chapter. The solution methods for different variants of the VRP are usually modifications of the methods proposed for the VRP. This chapter continues with the definition Heterogeneous VRP (HVRP) and details of the solution approaches presented by various authors.

Vehicle Routing Problem (VRP) is a well-known combinatorial optimization problem, where a number of customers, in different geographical locations are to be visited exactly by a vehicle that departs from a central depot and returns back to the depot. More formally, let $G = (V, E)$ be a graph where $V = \{v_0, \dots, v_n\}$ is the vertex set and $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the edge set. Vertex v_0 represents the depot. Each vertex corresponds to a customer with a non-negative demand. The aim is to find the least costly method that visits all the vertices using a specific edge set. According to Toth and Vigo (2001a), VRP generally have the following objectives:

- Minimization of the global transportation cost, dependent on the global distance traveled (or on the global travel time) and on the fixed costs associated with the used vehicles (and corresponding drivers);
- Minimization of the number of vehicles (or drivers) required to serve all the customers;
- Balancing of the routes, for travel time and vehicle load;
- Minimization of the penalties associated with partial service of the customers.

or any weighted combination of these objectives. VRP (more specifically Capacitated VRP, note that CVRP is used interchangeably with VRP throughout the section) is an NP-Hard problem (Lenstra, Rinooy Kan (1981)), thus there is no polynomial time complexity algorithm that can be used. Various solution methods have been proposed for the solution of the VRP. The literature on VRP is so wide that Toth and Vigo (2001a) report, 13 different survey papers, including the classification paper of Desrochers, Lenstra, and Savelsbergh (1990), the exact solution algorithms paper of Laporte and Nobert (1987), and the extensive bibliography by Laporte and Osman (1995). Moreover there are books on VRP by Toth and Vigo (2001a) and Golden and Assad (1988).

VRP has various extensions, like time windows, backhauls, capacity constraints, and vehicle number constraints. We are going to consider the generic VRP, or capacitated VRP (CVRP), as a great deal of solution methods have been experimented on this problem. Then, we more specifically concentrate on the HVRP. The more general studies that include extensions to HVRP, and the models including more than one depot and time windows (Dondo, 2003) are beyond of the scope of this study.

2.1. Solution Methods for the VRP

The methods reported in this sub-section are borrowed from the book of Toth and Vigo (2001a, 2001b). The solution methods can be grouped in three wide categories. The exact solution methods try to generate exact solution to the problem using mixed integer programming. The heuristic methods aim to generate good solutions using problem specific knowledge to generate good solutions. The metaheuristics are naturally inspired optimization mechanisms that try to improve the solutions based on problem specific heuristics.

2.1.1. Exact Solution Methods for VRP

Branch-and-bound and Branch-and-cut are two different approaches employed in solving VRP. Branch and bound method uses the branches of a search tree to construct a VRP solution. Relaxation of the subtour elimination or capacity cut constraints is used to in the early versions of branch and bound. Better relaxations

and good lower bounds are reported by Fischetti, Toth and Vigo (1994). Toth and Vigo (2001b) propose some promising branching schemes and search strategies.

Branch-and-cut approach is based on the relaxation of the integrality constraint in the mixed integer model (MIP) of the VRP. The MIP is solved as LP and the non integral values of the decision variables are used to generate the cuts. The branch-and-cut makes it possible to solve larger problems as only promising solutions are investigated by the cuts. Toth and Vigo (2001b) report that problems up to size of 45 customers can be solved to optimality with the branch-and-bound, while problems with 75 customers can be solved to optimality using the branch-and-cut method.

The set-covering based methods are also used in solving the VRP to optimality (Toth and Vigo, 2001b).

2.1.2. Heuristic Solution Methods for VRP

The heuristics for solution of VRP date back to 1964, (Clarke and Wright, 1964). There are various heuristic methods proposed since Clarke and Wright's Savings Heuristic. The heuristics are analyzed in three categories. The savings based heuristics, the two-phase heuristics and improvement heuristics.

Savings Based Heuristics

The Savings Heuristic proposed by Clarke and Wright (CW) (1964) is based on the notion of savings. When two routes are to be merged, the savings of the merger is calculated and a merger with the most savings is tried to be achieved. The saving (s_{ij}) for the merger of the following two routes: $(0, \dots, i, 0)$ $(0, j, \dots, 0)$ is $s_{ij} = c_{i0} + c_{j0} - c_{ij}$. There are various improvements on the classical CW heuristic. The complete enumeration of all possible savings is eliminated by Golden et al. (1988), while some authors showed that using only a fraction of c_{ij} when calculating the savings gives better results.

Desrochers and Verhoog (1989) and Altinkemer and Gavish (1991) showed that calculating the savings considering the best TSP tour before and after the merger of the two routes (Matching Based Savings (MBS) algorithm) gives better results

compared to the pure CW. Laporte and Semet (2001) report that the best saving based heuristic is devised by Wark and Holt (1994).

Moreover, there are some heuristics which insert the cities one by one, instead of merging tours, and these heuristics also make use of a measure for minimum increase in tour length that is quite similar to the savings calculations.

Two Phase Heuristics

The two phase heuristics include the route-first-cluster-second (RFCS) and cluster-first-route-second approaches (CFRS). The RFCS is based on constructing one giant tour and then partitioning this tour into routes. Laporte and Semet (2001) report that the solution quality of the RFCS is not as good as the CFRS. CFRS has attracted the operations researchers more, there are various clustering methods proposed by different authors. One of the most widely known ones is the sweep algorithm. The others are generalized assignment heuristic; location based heuristic, and the truncated branch and bound heuristic.

The sweep algorithm (Wren (1971) and Gillet and Miller (1974)) is a simple method. It is based on rotating a ray with infinite length starting from an arbitrary vertex and grouping the customers based on the angle observed when the customer intercepts with the ray (Laporte, Semet, 2001). The customers are included as long as the capacity allows, and the procedure is repeated until all the customers are grouped. A natural extension to the sweep algorithm is the Generalized Petal Algorithm, where several different routes are generated and the best route combination is found by solving a set partitioning problem. (Seyran (2006)).

In the generalized assignment heuristic (Fisher and Jaikumar (1981)), some (cluster) seed vertices are fixed at the beginning of the algorithms and the cost of inserting each customer to each cluster is calculated (Laporte and Semet (2001)). These costs and the customer demands are used to solve a Generalized Assignment Problem, to form the clusters.

According to Laporte and Semet (2001), the location-based heuristic is similar to the generalized assignment heuristic. Yet this time the seed vertices are generated by solving a capacitated location problem (Bramel and Simchi-Levi (1995)). The

remaining customers are then inserted so that the least insertion cost is incurred when TSP tours are generated.

The truncated branch-and-bound method is similar to the branch-and-bound methods used in the exact solution algorithms. Christofides et al. (1979) use the branches of the search tree containing only single branch and the levels consisting of non-dominated vehicle routes. The customers are gradually included in the routes according to the savings or insertion cost (Laporte and Semet (2001)).

Improvement Heuristics

The improvement heuristics are adopted from the Traveling Salesman Problem, which can be considered as a case of the single vehicle VRP. The tour constructed by the TSP heuristics is improved using some improvement heuristics that include edge or node recombination operations. A number of edges or customers are removed from the current route and reinserted in a place on the route that yields some improvement.

The improvement heuristics either can be limited to the single route or can exchange edges or customers within routes for a VRP. The comprehensive book of Reinelt (1996) includes various TSP improvement heuristics that can be employed when improving the routes in solving VRP. The most common improvement heuristics are apart from edge and node insertion are the 2-opt, 3-opt, and Lin-Kernighan (1973) approaches. In 2-opt, every edge is coupled with all other feasible edges to find an improvement when the tours are connected the other way around. The case in 3-opt is more complicated as three edges are deleted and there are 6 possible recombination moves. Or (1976) proposed a restricted 3-opt with a smaller time complexity generating good results. Lin and Kernighan (1973) propose an iterative heuristic where the λ -opt moves are not specified in advance and the algorithm terminates when there is no further improvement.

2.1.3. Metaheuristics for VRP

The metaheuristics are optimization methods trying to mimic the natural improvement mechanisms. Osman and Laporte (1996) state that “A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining

intelligently different concepts for exploring and exploiting the search space, and various learning strategies are used to structure information in order to find efficiently near optimal solutions.” Metaheuristics are not problem dependent and they can be applied to various problem domains by changing the subordinate heuristics. The descriptions of the main ideas of the heuristics are borrowed from Önder (2007), and the best performance is reported from a study of Gendreau et al. (2001).

Simulated Annealing

Simulated annealing (SA) proposed by Metropolis et al. (1953), is a metaheuristic based on statistical physics. The annealing process in physics seeks a good molecular structure by allowing formation of different molecular structures depending on the rate of change in the cooling temperature. SA is a search process controlled by a parameter, the temperature (Kirkpatrick et al. (1983) and Černý (1985)). The process is based on small changes in the current solution and the good moves are always accepted. When a move in an undesirable direction is encountered, the move is still accepted based on a probability depending on the temperature. At the initial phases of the algorithm, when the temperature is high, the algorithm accepts more non-improving moves. At the final stages when the temperature is gradually decreased, only improving moves are accepted. The algorithm explores the search space when the temperature is high, and exploits the current solution when the temperature is low. Simulated Annealing proposed by Osman (1993) obtained some of the best known results of problems with size 50 and 75 on instances of Christofides et al. (1979). The algorithm initializes using the CW to generate solutions and λ -interchange is applied for ($\lambda = 1$ or 2). Then simulated annealing is applied to the generated solution, small perturbations are made by λ -interchange operations. The cooling scheme is different from the classical point of view that the temperature is decreased when an improvement is found.

Deterministic Annealing

Deterministic Annealing (DA) follows the same idea with SA; however, the non-improving moves are accepted using a deterministic rule. The deterministic rule is

either calculated by adding a small perturbation factor to the current best solution (threshold-accepting) or allowing a small deviation, calculated as the fraction of the current solution (record-to-record travel). In record-to-record travel, a record is the best solution encountered during the search. Golden et al. (1998) used record-to-record travel on large instances generated by the authors and report two best-known results among 20 benchmark instances.

Tabu Search

Tabu Search (TS) is a deterministic search mechanism (with a limited memory) proposed by Glover (1986). The search is based on a hill climbing mechanism where memory is used to escape from the local optima. Hill climbing may get stuck at a local optima. Tabu search keeps in memory the points previously visited during the hill climbing mechanism as tabu points. Revisiting the tabu points is avoided to enforce the algorithm to explore the search space.

Practical experiences has shown that TS is the best metaheuristic proposed for VRP. Gendreau et al. (2001) report seven different TS applications with impressing results. Taillard's Algorithm (1993) gives the best results for 12 of the 14 Christofides et al. (1979) instances (Gendreau et al., 2001). The perturbations are based on λ -interchange moves and the individual routes are re-optimized after the exchanges. The author decomposes the problem into sub problems, partitioning the customers in sectors centered in the depot, and allowing the exchange of customers within sections.

Another TS algorithm that gives promising results is the "Taburoute" algorithm by Gendreau et al. (1994). The author uses the GENIUS heuristic developed for the TSP to perturb the current solution. The heuristic basically consists of two parts. GENI tries to insert the cities to the positions by evaluating the elimination of three new edges for each neighbor, within a p-neighborhood on a given tour; US tries to improve the tour by using reverse GENI operators. An advantage of the TS proposed by Gendreau et al. (1994) is that it allows infeasible intermediate points throughout the search process. The solution points that are infeasible in terms of vehicle capacity or maximal tour length are accepted after being penalized. The moves of

vertices among different vehicles are defined as tabu instead of whole solutions, and the tabu points are penalized instead of being disregarded from consideration. Taburoute obtained the 5 best known solutions of the 14 Christofides et al. (1979) instances (Gendreau et al., 2001).

Adaptive Memory Search is an interesting development of TS according to Gendreau et al. (2001). Rochat and Taillard (1995) proposed to use an adaptive memory to keep and dynamically update good solutions. The elements of different good solutions are selected and merged to give the resulting tours. The elements of the best solution are given a larger weight and repetition of a solution in the memory is avoided to avoid bias of the solution. The tours constructed from the resulting tour elements give good results and improve the best known solutions in two of the 14 Christofides et al. (1979) instances (Gendreau et al., 2001).

Genetic Algorithms

According to Larrañaga et al. (1999) Evolutionary Algorithms (EA) were proposed for solving probabilistic search problems by Bremermann et al. (1966) and Rechenberg (1973). Holland (1975) introduced the Genetic Algorithms (GAs) to optimization problems. GAs are based on “survival of the fittest” idea of Charles Darwin and genetic theory of Mendel. In GAs every solution is coded as a chromosome and the algorithm deals with a population of solutions instead of a single solution. These parent solutions are used to generate new children solutions that preserve chromosomes of previous ones. According to the schemata theorem (Holland, 1975) and building block hypothesis (Goldberg, 1989), the newly generated solutions preserve good characteristics of their ancestors, and the algorithm eventually converges to a good result. The GAs for VRP did not give promising results, the solution quality was worse than the simplest construction heuristics and the solution time was large (Gendreau et al., 2001). This observation is due to the lack of power of preserving good edges of the crossover operators used in the GAs. Crossover operators like the EAX (Nagata, Kobayashi, 1997) or NNX (Önder, 2007) that aim to preserve good edges in the parents can be helpful in generating good GAs. Moreover, the success of the GAs in the VRP with time-windows makes GAs a promising solution method.

Ant Colony Algorithms

Ant colony algorithm (ACA) proposed by Colormi et al. in 1991 (Gendreau et al, 2001). This algorithm represents the ant behavior to find the shortest route. When solving a TSP, a number of artificial ants move on a complete graph to find a route disposing pheromone, similar to the real ants. The pheromone evaporates with time, and the routes with highest pheromone levels are connected to give a tour. The main idea behind the scheme is that the short edges will have a higher level of pheromone as the ants will travel those edges in a shorter time. The approach is relatively new and lacks well established rules. There are very few studies that use ACA for VRP, yet the results of Bullnheimer et al. (1998) are comparable with other heuristics. ACA of Bullnheimer at al. (1998) is able to discover two of the best known solutions of the 14 instances generated by Christofides et al. (1979) (Gendreau et al., 2001).

Neural Networks

Artificial neural networks are based on neural activity model of Warren McCulloch and Walter Pitts (1943), mimicking central neural networks of animals (Michalewicz (2003)). The web of natural neurons does the reasoning in all animals. A neuron is a simple entity that gets inputs as a step function and reacts accordingly, multiplying the input signal and adding a preset weight. Elastic networks and self-organizing maps are quite remote from the classical artificial neural networks, but they have proven been to give better results (Gendreau et al., 2001). However, the results are not competitive with the alternative metaheuristics, in particular Tabu Search.

2.2. An Extension to HVRP and Solution Methods for HVRP

The heterogeneous VRP assumes that the vehicles can have different capacities for servicing the cutomers. The fixed capacity assumption of VRP is relaxed. The HVRP is more realistic than VRP as the vehicles in real life distribution fleets need not be homogeneous. Thus, the classical capacitated VRP is a special case of the HVRP, where the capacities of the vehicles are not the same.

The HVRP is studied by a limited number of researchers. This lack of interest can be explained with the difficulty of the problem (Yaman, (2006)). HVRP was first proposed by Golden et al. (1984). Two main approaches to the HVRP problem have arisen over time. In the first approach of Golden et al. (1984), it is assumed that the vehicles have a variable fixed cost, and the travel costs with different vehicle type assumed to be equal. On the other hand, Gendreau et al. (1999) assume that every type of vehicle has a different travel cost calculated based on the distance traveled. Various authors named the HVRP problem differently (Choi and Tcha (2005)). The approach of Golden et al. (1984) was named as: Vehicle Fleet Mix (Salhi, Rand, 1993), Fleet Size and Mix VRP (Golden et al. (1984)), Fleet Size and Composition VRP (Gheysens et al. (1986)), and HVRP (Yaman (2006)). The approach in which the variable travel costs are considered has been referred to: HVRP (Gendreau et al., 1999), VFM with variable unit running costs (Salhi et al., 1992), the mix fleet VRP (Wassan, Osman, 2002). We will refer the method in which the travel costs are assumed constant as the Vehicle Fleet Mix (VFM), and the second method as the HVRP throughout in this thesis, the HVRP will also be used in a broader sense when the problem class is referred.

Salhi and Rand (1993) present a detailed literature review of the early solution approaches to the HVRP. According to the authors, linear programming models generated for the problem dates back to 1969, when Gould (1969) generated a linear program for the non-homogeneous fleet. There are about twelve different studies mentioned by Salhi and Rand (1993) related to the HVRP, who tried to solve the problem using analytic, statistical, or stochastic methods. The work of Golden et al. (1984) presented solution approaches for real life problems by the heuristic approaches. Yaman (2006) states that there is no exact solution method proposed, the only study related to the exact solution consists of the lower bounds calculation of Golden et al.(1984).

Yaman (2006) presents six different formulations of the VFP, four based on vehicle flow and two based on flow of variables. Miller-Tucker-Zemlin (1960) constraints are used in subtour elimination and the author states that the lower bounds are improved using valid inequalities. LP relaxations of the Golden et al. (1984) instances are studied and lower bounds for the problems are derived. The

formulation and lifting equalities of Yaman (2006) are used in Chapter 3 to derive a lower bound for the real life problem with 90 customers. Yaman (2006) reports that the maximum percentage gap between the lower bound and the upper bound for the problems (up to 100 customers) is 4.9%. The gap is desired to be as small as possible as the performance of the heuristic proposed will be assessed by the calculated lower bound.

Golden et al. (1984) were the first to develop heuristics for the VFP, based on the CW heuristic on tour partitioning. The authors proposed four new saving calculations that can be used to merge partitioned tours. The first improvement was to add the vehicle costs in the savings calculations. The savings calculation is based on the opportunity cost that arises when a vehicle with too large capacity is used in order to combine two small routes; cost of the unused capacity is subsidized by deducting the cost of the vehicle with smallest capacity from the savings to favor the merger of tours. The third calculation takes into the consideration of the unused capacity and deducts the cost of the largest vehicle, which can accommodate for the unused capacity, from the savings calculation. The last formulation is a parametric formulation in which the cost of the newly formed edge deducted by a factor of $(1-\gamma)$, where γ is the route shape parameter.

Golden et al. (1984) also proposed the use of a RFCS technique to solve the VFP. A tour including all the customers, excluding the depot is constructed and then partitioned to small tour so that the length of the newly added edges is minimized. The partitions do not always results in good solutions thus the procedure is repeated for five times and the resulting tours are subjected to 2-opt or Or-opt (Or, 1976) moves.

Gheysens et al. (1984) used the lower bounding principle to generate the solution to the VFP. The cluster-first-route-second approach is used to generate the solution. The LP relaxation to the problem is initially solved and the fleet composition of the LP relaxation is used with generalized assignment heuristic.

Desrochers and Verhoog (1991) used the MBS algorithm to calculate the savings proposed by Golden et al. (1984). They used a TSP solver proposed by Jonker and

Volgenant (1984). The authors demonstrated that the VFP can be solved using a good TSP solver with relatively good results. However, the authors conclude that the best result is obtained by the last version of the savings algorithm proposed by Golden et al. (1984) or the partitioning algorithm is combined with the Or-opt. Moreover, the procedure proposed by Gheyens et al. (1984) was consistent in producing good results with the settings of Desrochers and Verhoog (1991).

Salhi and Rand (1993) developed a group of different perturbation mechanisms that can be used to improve the routes of a given VFP. The methods they propose are based on edge and node exchanges combined in an iterative manner. The perturbation methods proposed are capable of merging small tours, exchanging customers between tours and partitioning large tours into small tours. The results are compared with the partitioning and Or-opt method, LB and routing method and the MBS method by the previous authors. The heuristic proposed by Salhi and Rand (1993) produced better results in terms of percentage deviation and was able to discover the best known results for 8 of the 20 test instances.

Osman and Salhi (1994) propose a tabu search application, in which the initial solution is generated using the method proposed by Salhi and Rand (1993), and the λ -interchange mechanism. The authors improved the best known results of 7 of the 20 problem instances of Golden et al. (1984).

Gendreau et al. (1999) used a tabu search using the GENIUS within an adaptive memory search. The algorithm uses a sweep-based method in selecting the edges that are to be inserted, by rotating a ray to select the cities. Tabu search limits the number of customers to be moved by GENIUS movements. The routes are subject to post optimization that is based on movements like 2-opt and merger of smaller routes to larger routes that can be served by a larger vehicle. Adaptive memory search is used to generate the final tours, using the best tours in the memory. The authors improved the results of 8 of the 12 benchmark instances of Golden et al. (1984).

Taillard (1996) used the column generation method to generate good solution for the adaptive memory search for the HVRP. The author proposed to use a generic LP

solver to solve the set partitioning problem that arises when the vehicle capacities are assumed equal. The results of the column generation attempts with different capacities are memorized in the adaptive memory search. The final HVRP tours are constructed using the information stored in the adaptive memory. The author improves the results of all the studies mentioned, for the Euclidean problems. Taillard (1996) also proposes measuring the solution quality without considering the fixed vehicle cost, only using the traveling cost arising due to the cost differences of using different vehicle types. This version of the HVRP will be referred as HVP, throughout the thesis.

Ochi et al. (1998a) proposed the use of GA that incorporates problem specific knowledge. The initial solution for the GA generated using sweep technique. The ERX crossover operator proposed by Whitley et al. (1989) is employed. The crossover operator is one of the early applications of operators that aim to preserve the good edges present in the parents. Tours are randomly perturbed, by a mutation operator that randomly changes the positions of random length strings in the tour representations within vehicles. The authors do not report the results in tables, thus the quality of the solution cannot be judged easily from the graph plotted. Ochi et al. (1998b) report improvements in the solution time and quality when parallel populations are used. The results are again presented in graphical format, thus it is hard to assess the quality of the solutions.

Renaud and Boctor (2002) proposed a sweep based algorithm which is followed by some improvement moves. The authors first define an order for the edges to be swept for both the Euclidean and non-Euclidean cases. Then 1 and 2 petal routes are generated. An eleven step improvement procedure which mostly consists of 2-opt and 3-opt moves is applied. The algorithm gives the best results compared to the VFM heuristics. The results of the sweep are also compared to those of the tabu search applications, and the results are better than the application of Osman and Salhi (1994) on the average, and slightly behind the results of Taillard (1996) and Gendreau et al. (1999). Moreover, the algorithm is capable of generating solution for the non-Euclidean distances, which was not considered by Taillard (1996) and Gendreau et al. (1999).

Lima et al. (2004) proposes the use of GA that incorporates problem specific knowledge. The initial solution used by the GA is generated using the GENIUS heuristic and the individuals are subjected to λ -interchange moves ($\lambda = 1, 2$). The algorithm uses the ERX as the crossover operator, again yet the λ -interchange is much powerful compared to the random string exchange. The algorithm generated new best solution for 8 of the 20 problem instances.

Choi and Tcha (2005) used the column generation method to solve the VFM and HVRP. The method proposed is based on the LP relaxation of the initial model based on the formulation using the Danzig-Wolfe decomposition. The algorithm is based on the simplex solution, generating the columns when it is necessary using the dynamic programming approach, and solving the resulting problem using the branch and bound method. The results are guaranteed to be optimal due to the partial column generation, yet the authors report that the absolute majority of the solutions generated are near optimal. The authors report that the algorithm is capable of discovering the best-known value for 11 out of 12 VFP instances and 5 out of 8 HVP instances. The deviations from the best knows' are 0.004 % and 0.015 % for VFP and HVP respectively.

CHAPTER 3

A MATHEMATICAL MODEL FOR EMPLOYEE TRASPORTING PROBLEM

The problem of the biscuit producing firm is formulated and modeled as a mixed integer model in this chapter. The model is basically a routing model with side constraints and extensions specific to the firm. The personnel that are to be carried to the production plant shall be brought to the plant within a specified period of time, and the vehicles used by the firm are not identical. The model we have developed thus is based on the Heterogeneous Vehicle Routing Problem (HVRP).

This chapter continues with the statement and formulation of the problem in detail, followed by the assumptions adopted while generating the mathematical model described. The constraints of the mathematical model are described in detail next. The chapter continues with the preliminary runs conducted on benchmark problems. The results obtained after the preliminary runs and the lower bounds proposed are summarized at the end of the chapter.

3.1 Statement and Formulation of the Vehicle Routing Problem

The problem is to carry the employees of a firm to the plant using a number of busses. The passengers to be collected or delivered are spread throughout different bus stops of a city. All the busses are to start their journey at the plant and bring all the employees to the plant.

The aim is to minimize the fixed cost of the number of busses used. The firm has a subcontracting agreement, due to which the firm only pays a fixed cost to the subcontractor based on the bus types used.

The fixed cost of the vehicles is calculated by multiplying a fixed cost by the number of passenger places present in a given bus type. There are two types of busses currently available at the subcontractor, one with a capacity of 20 and one with capacity of 40 passengers. The cost of two small buses is equal to the cost of one large bus.

The decision to be made is the number of busses and the routes each bus will follow. The route is important as there is an implicit restriction on the route length due to the policies of the firm. The time between the pick-up of the first passenger and arrival at the plant is limited to be less than an upper limit, 75 minutes. The firm states that it is desirable to have this time as short as possible. On the other hand this time limit is not strict; this time limit can be exceeded in extreme conditions, depending on the cost that will be incurred if the limit is kept constant.

Thus the model that is described in Section 3.3 has an objective function that uses goal programming to incorporate the time restriction specified by the firm management.

The capacities of the busses cannot be exceeded in any case, thus overloading is not possible. Moreover, all the bus stops are to be visited, the firm does not have to possibility to handle the passengers by other means.

3.2 Assumptions of the Routing Problem

The assumptions employed while constructing the mathematical model are described in this section. The assumptions are based on the policies of the firm and the basic assumptions of the CVRP.

- Capacity of each bus is known prior to the solution
- All passengers are identical in terms of capacity usage in the busses
- A bus is assumed to pick-up all the passengers waiting in a bus stop: This assumption is due to the policy stated by the subcontractor. This is made possible by making small changes in the places of the bus stops if the solution requires. The firm is able to make changes, like splits of the bus stops in two, and placing the bus stops so that all the passengers in a bus-stop are picked up by the bus that visits a given bus stop.

- The travel time is proportional to the route length: In a case study of school bus routing, Bramel and Julien (1997) state that the travel time calculated using a fixed speed and the passenger pick-up times can be modeled using linear regression with a p value smaller than 0.001. The authors however point out that using a universal speed for different levels of congestion is not adequate. The fixed speed shall be modified according to the congestion level.
- The current bus stops used are assumed as fixed: The selection of the location of the bus stops, thus making splits in the demands at the bus stops is beyond the scope of this study. The decision of bus-stops is not limited only to the splitting of the bus stops and also includes the decision of locations.
- The bus stops with maximum number of passengers has less passengers than the capacities of the bus with the least capacity
- The number of people waiting at each bus stop is known
- Each bus can do only one trip in each planning horizon
- Cost of the busses is proportional to the capacity of the buses
- The plant works in three shifts, thus a bus shall start and finish its tours at the plant.

3.3 Mathematical Model of the Routing Problem

The mathematical model we have developed for the solution of the routing problem is based on Miller-Tucker-Zemlin (1960) constraints. The subtour elimination constraints of the proposed by the authors is used with the capacity constraints proposed by Golden et al. (1984), extended by Yaman (2006).

We have developed the objective function and the route length constraints. The objective is to minimize the total fixed cost of vehicles, and minimizing the weighted tardiness on the routes. As the lateness is not desired at all, a management policy that shall be incorporated.

All the constraints are described within this section, after the statement of the notations and the decision variables.

N : Set of nodes in the problem

M : Very large number

T : Maximum allowed time

A : Graph that includes all the arcs between N nodes

AK : Graph containing all the combinations of arcs in A using vehicle types of K

Q_k : capacity of vehicle k

K_i : the set of vehicles that can serve node i

d_{ij} : shortest distance from i to j ($(i, j) \in A$)

C_{ik} : fixed cost of making a trip using vehicle type k

L_i^+ : Length of route that terminates between node i that is larger than the allowable route length

L_i^- : Difference between the maximum tour length and the tour length not used by the route that terminates at node i

w_T : time per kilometer traversed (1/fixed speed)

w^+ : weight per kilometer of road that exceeds the time limits

w^- : weight per kilometer of road that improves the time limits for route

u_i : total demand on the trip until i

t_i : total distance elapsed until node i is visited

$$a_{ik} : \begin{cases} 1 & \text{if there is a trip that uses vehicle type of } k \in K_i \text{ that ends at node } i \\ 0 & \text{otherwise} \end{cases}$$

$$b_{ik} : \begin{cases} 1 & \text{if vehicle type of } k \in K_i \text{ that goes through node } i \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} \text{if arc } (i,j) \in A \text{ is used using vehicle type } K [(i, j, k) \in AK] \\ 0 & \text{otherwise} \end{cases}$$

The mathematical model we propose is:

$$\min \sum_{i \in N} \sum_{k \in K_i} C_{ik} a_{ik} + w^+ \sum_{i \in N} L_i^+$$

s.t.

$$t_i + d_{ij} x_{ijk} - t_j \leq M(1 - \sum_{k \in K} x_{ijk}) \quad \forall (i, j) \in A \quad (3.1)$$

$$w_t(t_i + d_{io}) \leq T \sum_{k \in K} a_{ik} + L_i^+ - L_i^- + M(1 - \sum_{k \in K} a_{ik}) \quad \forall i \in N \quad (3.2)$$

$$L_i^+ + L_i^- \leq M \sum_{k \in K} a_{ik} \quad \forall i \in N \quad (3.3)$$

$$\sum_{j: (0, j, k) \in AK} x_{0jk} = \sum_{i \in N: k \in K_i} a_{ik} \quad \forall k \in K \quad (3.4)$$

$$\sum_{j: (j, i, k) \in AK} x_{jik} = a_{ik} + b_{ik} \quad \forall i \in N, k \in K \quad (3.5)$$

$$\sum_{j: (i, j, k) \in AK} x_{ijk} = b_{ik} \quad \forall i \in N, k \in K_i \quad (3.6)$$

$$\sum_{k \in K_i} (a_{ik} + b_{ik}) = 1 \quad \forall i \in N \quad (3.7)$$

$$u_j \geq u_i + q_j - \sum_{k \in K_i} Q_k (a_{ik} + b_{ik}) + \sum_{k \in K_{ij}} Q_k x_{ijk} \quad \forall (i, j) \in A: i \neq 0 \quad (3.8)$$

$$u_i \geq q_i + \sum_{k \in K} \sum_{j: (j, i, k) \in AK} q_j x_{jik} \quad \forall i \in N \quad (3.9)$$

$$u_i \leq \sum_{k \in K_i} Q_k (a_{ik} + b_{ik}) \quad \forall i \in N \quad (3.10)$$

$$a_{ik} \in \{0, 1\} \quad \forall i \in N, k \in K_i \quad (3.11)$$

$$b_{ik} \in \{0, 1\} \quad \forall i \in N, k \in K_i \quad (3.12)$$

$$y_{ijk} \in \{0, 1\} \quad \forall (i, j, k) \in AK \quad (3.13)$$

The first two constraint sets impose the time constraints to the model and calculate the earliness and tardiness. The first constraint set calculates the total tour length encountered at node i , for the vehicle, this distance is used in the second and third constraint sets to calculate the earliness of tardiness of the vehicles. A constraint in the first set becomes active only if there is a trip that uses the arc $i - j$, calculating the time of visit of node j . The second constraint incorporates the last trip, from the last node (bus stop) in a route to the depot (plant). The tardiness is calculated using

this constraint in second set. Similar to the first set, a constraint becomes active only if there is a trip terminates after node i is visited for any vehicle type. The third constraint set, assures that the tardiness or earliness are calculated only if there is a routes that terminate at node i .

The constraints four through ten are proposed by Golden et al. (1984), and enhanced by Yaman (2006). The definitions of the constraints are borrowed from Yaman (2006). The fourth constraint set assures that the number of vehicles leaving the depot is equal to the number of the routes that terminate at the depot. The fifth constraint set assures that there is an incoming arc to every node which is served by a vehicle of type k . The sixth constraint set states that there is an outgoing arc from every node that is visited by vehicle type k and does not end a trip at node i . The seventh constraint set ensures that each node is visited by one type of vehicle only, and a node is either passed through or is a terminating node of a route.

The eighth constraint set linearizes the constraint $u_j \geq (u_i + q_j) \sum_{k \in K_{ij}} x_{ijk}$, if for some k , $x_{ijk} = 1$, as $a_{ik} + b_{ik} = 1$ then $u_j \geq u_i + q_j$. If $x_{ijk} = 0$, then $u_j \geq q_j$ and $u_i \leq \sum_{k \in K_{ij}} Q_k (a_{ik} + b_{ik})$.

The ninth constraint set replaces the weaker constraint $u_i \geq q_i$ by Desrochers and Laporte (1991), improved by Yaman (2006). The utilization of a vehicle that has visited node i after visiting a series of nodes lasting with j , updates such that the demand of i is added to the previous utilization value. The tenth constraint set ensures that the sum of demands of nodes on trip that goes through node i is less than or equal to the capacity of vehicle type k .

3.4 Initial Experiments

The mathematical model described above is compiled using GAMS 2.0 on a PC with AMD Tution 64x2 1.6 GHz CPU, 512 MB of RAM. The problems defined by Golden et al. (1984) are used to test the performance of the mathematical model running built in CPLEX solver. As mentioned in the previous chapter, the set of problems defined by Golden et al. (1984), consist of the problem set that has been most widely studied. This problem set has best-known solutions for larger problems

that cannot be solved to optimality using PCs. The availability of good solutions for problems with size as large as 101, is adequate for experimenting as the problem of the firm we are in contact with is 92.

The problems are to be modified for our model as the problems proposed are HVRP problems and the aim is to minimize the total fixed and routing costs without any time limitations. The fixed costs proposed by Golden et al. (1984) are used in the modified version we have used. The time limit is assumed to be 100 for modified the problems, although Lui and Shen (1999) proposed using the length as 500, in their study on Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW). HVRPTW is out of scope of this study and the results are not comparable, yet the authors have demonstrated that this time windows are tight enough to limit find good solutions that are not exactly the same with the HVRP solutions. The length is reduced as the tour length in smaller problems is hardly above 500, and the aim of our model is to find a point that balances the tardiness fixed cost. Lui and Shen (1999) proposed three different weight levels for w_T as 1.1, 1.30 and 1.50. We assumed w_T as 1, yet w^+ is used as 1.1 and 1.5.

Table 3.1 summarizes the results and the CPU times on PCs* with Pentium 4, 2.4 GHz having 256 MB of RAM. The smallest problem proposed by Golden et al. (1984) has a size of 13, thus the problem size is reduced by deleting some nodes randomly to test the performance of the model. Vehicle capacities are not modified to see the impact of increase in the number of customers on the solution time.

Table 3.1 Results of the mathematical model and CPU times

Size	$w_T=1.1$		$w_T=1.5$	
	Opt	CPU Time (seconds)	Opt	CPU Time (seconds)
6				
7	275.63	6.03	275.63	6.03
8	270.00	60.70	369.94	135.47
9	324.00	817.84	324.00	777.66
10	335.89	7012.47	340.20	6735.30
11*	450.00	20380.59	450.00	20424.63
12*	486.30	30294.13	487.50	28965.28
13*	-	30827.42	-	27954.20

*Values are the best integer solution observed, not necessarily optimal

* Computer Aided Manufacturing and Design Laboratory of Çankaya University Industrial Engineering Department was used for the experimental runs.

The problem having number of customers larger than 10 could not be solved to optimality due to lack of memory. The results indicated in the table for sizes 11 and 12 are the best integer solution obtained after the mentioned run time. No feasible integer solution is found during the run time of 8.5 hours before the memory limits are exceeded for the original problem 1 of Golden et al. (1984)

3.5 Results of the Initial Experiments

The Table 3.1 clearly identifies that the problems with relatively large size, larger than 10 cannot be solved in reasonable amount of time. The exact solution with the given formulation is not possible for problems with size higher than 10.

The following section will continue with solution methods we have developed based on some promising applications that are present in the HVRP literature. The problem of assigning the absolute quality of these solution methods is to be overcome by testing these methods on pure HVRP instances. The best-know solutions are applicable to our solution methods.

The heuristic and metaheuristic methods we apply in the following section are coded such that, the time restrictions and the objective function can be modified easily. The objective function and the time restriction constraints are eliminated when testing the algorithm with the pure HVRP instances. After analysis of the performance of the algorithms on pure HVRP instances, the objective function and the time restrictions are incorporated.

On the other hand the solution quality can also be measured based on the linear relaxation of the problem, which takes relatively very small amount of time. The following subsection includes the linear relaxations and the results of these formulations.

3.6 Linear Relaxation of the Mathematical Model

The mathematical model developed is solved assuming some of the binary variables continuous to find some lowed bounds. The relative gap is reported to be a good measure by various authors including Yaman (2006) and Choi and Tcha (2005). The relative gap is defined as $\left(\frac{UB - LB}{UB}\right) \times 100$, where UB stands for upper bound and

LB stands for the lower bound. This measure is used to assess the relative deviation of a calculated solution (an upper bound).

Yaman (2006) proposes some valid inequalities for HVRP to reduce the percentage gaps for different formulations of the problem. The following two valid inequalities are implemented as proposed by Yaman (2006). (14) is reported to improve the relative gap over 75% for instances 3-6 proposed by Golden et al.(1984).

$$\sum_{i \in N} \sum_{k \in K} \left\lfloor \frac{Q_k}{Q} \right\rfloor a_{ik} \geq \left\lfloor \frac{\sum_{i \in N} q_i}{Q} \right\rfloor \quad (3.14)$$

The equation is valid for all values of Q , and eliminates some facets on the search space. The constraint ensures that the total capacity of vehicles used is less than the total demand. Yaman (2006), proposes to use Q as the greatest common divisor of all vehicle capacities to obtain the best facet cut, in order to decrease the relative gap.

(15) is reported to improve at least 4.15%, thus this inequality is also used as a valid inequality to improve the relative gap. This inequality can result in improvements as large as 18.78%.

$$Q_k \sum_{i \in N} \sum_{k \in K} a_{ik} \geq \sum_{i \in N} \sum_{k \in K} q_i (a_{ik} + b_{ik}) \quad (3.15)$$

This inequality ensures that the sum of demand satisfied by vehicles that either pass through a node or terminate at a node is less than the total capacity of vehicles. The constraint is obviously redundant in the MIP formulation.

Other inequalities based on reduction of the sub-tour elimination moves on the search space are not included in our lower bound calculation; as the large amount of sub-tour elimination constraints does not rationalize the time spend in implementing the constraints. The process can be demonstrated better by a solid example; say for problem instance 13 of Golden et al. (1984), when all inequalities and lifting equations are implemented the relative gap is 12.02%. On the other hand when the a_{ik} and b_{ik} are kept as binary in the linear relaxation, the relative gap is 10.47%, while the relative gap is 14.03% when all variables are assumed continuous.

The lower bounds for all instances that are used in this study are calculated using the mathematical model discussed in Section 3.3, loosening the integrality constraint, assuming variables x_{ijk} continuous. The lower bounds for our model assume all variables as continuous.

Two different lower bounds are calculated for each problem. The one of the lower bounds (denoted as LB) corresponds to the relaxation of our model, the second corresponds to the linear relaxation of the pure HVRP problem (denoted as LB(HVRP)), which aims to minimize the vehicle and travel cost. The second lower bound uses the inequalities (14) and (15), which are proven to improve the percentage gap. These second lowers bound can be used to asses the relative quality of the solutions that is comparable to similar results in the literature. Note that LB for our model are much looser than the LB for HVRP as all variables are assumed continuous and no valid equalities are included. The inequalities need to be proved when time restriction constraints are added.

Table 3.2 Lower bounds for randomly located problem instances, problem set 1

	LB	LB(HVRP)
3	373.04	941.85
4	3840.91	6402.54 [*]
5	390.00	844.19
6	3840.91	6318.17
13	1061.10	2156.15
14	6432.43	8974.45
15	995.89	2198.66
16	1950.00	2556.65 [†]
17	701.10	1468.77
18	775.00	2179.15 [†]
19	7300.00	8113.08 [†]
20	2500.00	3546.07

These lower bounds are used throughout the following chapters in calculating the relative gaps for all problems. Two sets of problems are considered, the problems proposed by Golden et al (1984), forms the first set. The original numbering structure of the authors followed, the problems that are not present are non Euclidian instances and are not considered in the scope of this study as heuristics are based on the locations of the points. Table 3.2 summarizes the lower bounds. It shall be

^{*} This lower bound is better than the lower bound (6369.15) proposed by Choi and Tcha (2005) in their work on linear programming relaxation that has been solved using column generation technique.

[†] $a(i,j)$ and $b(i,j)$ are also assumed as continuous for generating the LB of these instances

mentioned that the instances are generated such that effects of different price structures can be demonstrated. For instance, problem instances 3 and 4 are identical in terms of node placement, yet the vehicles used in problem 3 are much expensive then the price per capacity is considered. If it costs 1 with the smallest vehicle in the first price structure, it costs 16.6 units to carry the same load with the smallest vehicle in the second price structure. The similar price structures are used in all problems of Golden et al (1984).

A drawback of the instances used in the first set is that the nodes are distributed evenly throughout the graph. To asses the performance of the heuristics that are implemented in the next chapter, clustered instances are used. The clustered instances proposed by Solomon (1987) as VRPTW are modified to become HVRP. The instances of Solomon (1987) are basically modification of delivery times of two different cluster structures, in our implementation the clusters are preserved as proposed by the author and the vehicle capacities are used as proposed by Golden et al. (1984). Thus each problem of Solomon (1987) is modified to be served by two different vehicle fleets, a cheap and an expensive fleet, the lowerbounds can be seen in Table 3.3. So that, the performance of the heuristics applied can be analyzed better in this controlled environment.

Table 3.3 Lower bounds for clustered problem instances, problem set 2

	LB	LB(HVRP)
1	201.41	803.34
2	3840.91	6209.40
3	205.00	899.16
4	3840.91	6269.72
5	1100.00	2435.93
6	6432.43	8845.48
7	995.89	2443.81
8	1156.08	2520.78*
9	701.10	1312.81
10	821.10	2096.42*
11	7300.00	7989.55*
12	2700.00	3821.84

* $a(i,j)$ and $b(i,j)$ are also assumed as continuous for generating the LB of these instances

CHAPTER 4

IMPLEMENTED ALGORITHMS

This chapter briefly describes different solution approaches implemented to solve the HVRP. The algorithms consist of heuristics and metaheuristics to solve the HVRP and HFFVRP, in a reasonable amount of time, close to an optimal solution. Five different solution approaches are presented in the chapter. The first two approaches are based on the decomposition heuristics, RFCS and CFRS are applied to HVRP. The routing phase in both of the solution methods are solved to optimality, yet to assure optimality the clustering and routing are to be done simultaneously, the best solutions are not necessarily optimal. The first metaheuristic to be used is a genetic algorithm that aims to preserve the edges in the parents. The algorithm uses a crossover and a mutation operator based on conventional heuristics. The search is intensified using simple improvement heuristics, yet moves are limited using two types of memory structure. The last solution approach is based on agent systems; this study will include the first application of agent systems to HVRP.

This chapter includes the algorithms of the solutions implemented on two benchmark sets defined in the previous chapter; the results of the implementations and the comparison in between the algorithms are given in Chapter 5.

4.1 Route-first-cluster-second Algorithm for HVRP

The ideas of Golden et al. (1984) and Gheysens et al. (1984) are followed in this algorithm with major improvements. Authors presented their ideas when the theoretical background about the solution of TSP was not very deep. In this method,

we consider finding an optimal solution to the giant TSP tour that consists of all of the customers then using an efficient clustering heuristic based on mathematical programming is considered.

TSP solving algorithms use available software online; one can solve instances of size 100 within seconds. Largest problem contained in the test instances proposed by Golden et al. (1984) consists of 100 customers. Önder (2007) notes that, problems containing 85,900 cities are solved to optimality (Reinelt (2007)) today. According to Reinelt (1996), the progress in the ability to solve the problems with large sizes “is only partly due to the increase in the hardware power of computers. Above all, it was made possible by the development of mathematical theory (in particular combinatorics) and of efficient algorithms” (Önder, 2007). However, TSP cannot be considered easy to solve, as the complexity of the problem increases exponentially with the number of cities. TSP is a member of NP-hard problems. Therefore, an efficient and effective solution procedure is required. Currently, CONCORDE (Cook, 2007) is a powerful tool for generating exact solutions for small and medium sized problems and lower bounds for problems as large as 1,904,711 cities (Applegate, 2007).

The routing phase of our first algorithm is done using the CONCORDE (Cook, 2007). The CONCORDE solves the TSP problem using a highly specialized branch-and-cut algorithm. Three different cutting algorithms are applied to improve the separation for subtour cuts (Applegate et al., 1998). The authors add that subproblems are solved using branch-and-cut method, improving the data structures. Problem instances with size up to 2392 have been solved with only one node, in 438.9 seconds, where a problem of size 120 is solved in 3.3 seconds.

We have formulated the clustering using a mixed integer model in the second phase of the algorithm, using the tour obtained from the CONCORDE solution. The model is designed to assign cities in the order obtained by the first phase, where the decision made is the assignment of each city to one of the vehicles, and the capacity of the vehicle. The notations and the decision variables that are used in the model are:

i = ordered set of the customers obtained by solving the TSP

k = number of vehicles utilized by the model

t = types of vehicles that can be used in the model

c_{i0} = distance between customer i and the depot

f_t = fixed cost of vehicle k

Q_t = capacity of vehicle type t

q_i = demand of customer i

d_i = distance incurred to finalize a subtour and start the next (following) tour from city i

$$x_{ik} = \begin{cases} 1 & \text{if city } i \text{ is assigned to vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{kt} = \begin{cases} 1 & \text{if vehicle } k \text{ is of type } t \\ 0 & \text{otherwise} \end{cases}$$

The mathematical model we propose for clustering is given as follows:

$$\text{Min } \sum_{k=0}^K \sum_{t=1}^T f_t y_{kt} + \sum_{i=0}^I d_{i-1}$$

s.t.

$$(c_{i0} + c_{(i-1)0}) \sum_{k=0}^K \left(x_{ik} - \sum_{k \neq m} x_{(i-1)m} \right) \leq d_{i-1} \quad \text{for all } i = 1, \dots, I \quad (4.1)$$

$$\sum_{i=1}^I q_i x_{ik} \leq \sum_{t=1}^T Q_t y_{kt} \quad \text{for all } k = 0, \dots, K \quad (4.2)$$

$$\sum_{t=1}^T y_{kt} = 1 \quad \text{for all } k = 0, \dots, K \quad (4.3)$$

$$\sum_{k=1}^K x_{ik} = 1 \quad \text{for all } i = 1, \dots, I \quad (4.4)$$

$$\begin{aligned} x_{ik}, y_{kt} &\in \{0,1\} \\ d_i &\geq 0 \end{aligned} \quad \text{for all } i = 1, \dots, I \quad (4.5)$$

The objective is to minimize the total fixed cost incurred, when the vehicle types are assigned to the routes, and the distances traveled between the first and last nodes assigned to each vehicle. The model contains four types constraints, and is an extension of the set partitioning model to our problem. The first constraint calculates the distances incurred when a tour is finished and the next one is initiated. The

constraint adds the sum of the distances of between the last customer of a tour and the depot and the distance between the first customer of the next tour and the depot. The second constraint set limits the number of customers assigned to a vehicle; the constraint limits the capacity of the vehicle enforcing one of the capacities available. The third constraint set ensures that the vehicles are limited to only one capacity; the last constraint ensures that each customer is visited by only one vehicle.

The model was run for all problems proposed by Golden et al. (1984), where a giant tour was formed using CONCORDE (Cook, 2007). The model was compiled using GAMS 2.0 on a PC with AMD Tution 64x2 1.6 GHz CPU, 512 MB of RAM. (GAMS code can be seen in Appendix C) The mathematical model was run on PCs with 256 BM of memory using built-in CPLEX solver, and all of the models terminated due to lack of memory. As heuristic clustering methods are reported to give solutions that are not comparable to CFRS methods (Laporte and Semet (2001)), this method is discarded from further consideration.

4.2 Cluster-first-route-second Algorithm for HVRP

Renauld and Boctor (2002) demonstrated that CFRS approaches successfully generate good results for HVRP. A sweep algorithm similar to the one applied by the authors is implemented in this solution approach. Moreover, Gencer et al. (2006) used a parameter to decide on the cluster size in HFVRP problems. The algorithm used also makes use of a parameter similar to the proposed by Gencer et al. (2006) that is a measure of wideness of the clusters.

The sweep algorithm is used for clustering the cities, a ray with infinite that is centered at the depot is rotated on the Euclidian plane, and the cities are included in the order they appear. The polar coordinates are used to explain this process the customers with least polar angle, θ is selected first. The process is based on the observation that the best known solution are observed in petals, although some petals intercept.

In our implementation of the sweep algorithm we used a measure named as radius of convergence ρ , which is defined by a shape parameter (R) multiplied by inverse of capacity utilization and radius of coverage. The radius of coverage (σ) is defined as

the radius of the smallest circle that can encapsulate all the customers assigned to a vehicle.

$$\rho = R \frac{Q_{[i]}}{Q(S)} \sigma$$

The center used for calculation of the σ is an important decision measure. The center is calculated using the weighted distances, the demand of each customer is used as a weight factor as it is advantageous to include customers with large demand that are not too distant to each other. A modified version is used to calculate σ such that the depot is also included. The weight of the depot is assumed to be equal to sum of the demand of all customers assigned to that cluster. Gencer et al. (2006) report that the customers that are near to the depot shall be considered while doing the routing as a vehicle that travels a great distance can serve the customers that are near the depot as long as it has free capacity.

The algorithm is initiated with the vehicle having minimum capacity, and if the capacity required to handle the customers assigned by the sweep move is less than the available capacity the inner loop terminates and the customers are assigned to that vehicle. However, if the capacity required to handle the customers is larger than the capacity of the current type of vehicle, then three different methods are used. There is no best method defined in the literature to select the type of vehicle, thus we have used all three methods proposed by different authors for selection of the vehicle type. The first method is proposed by Renaud and Boctor (2002) for sweep algorithms is reported to give promising results. The method that is proposed by Ochi et al. (1998a) was implemented on genetic algorithms, and the last method is deciding on the vehicle type randomly each time the capacity of a vehicle is to be exceeded. Figure 4.1 demonstrates the pseudo code of the algorithm when the Renaud and Boctor (2002) measure is used.

Renaud and Boctor (2002) compare the costs of using two different vehicles, and using single larger vehicle. Assume that S denotes a set of customers assigned by the sweep algorithm. $Q_{[1]}$ denotes the capacity vehicle with least cost, and the $Q_{[i]}$ denotes capacities of larger vehicles in an increasing order. For the case $Q(S) \geq Q_{[i]}$,

then cost of assigning $Q(S)$ to $Q_{[i+1]}$, and comparing cost of $Q_{[1]}$ and assigning $Q(S)$ - $Q_{[i]}$ to $Q_{[1]}$ (Renauld and Boctor, 2002). The capacity of the vehicle is increased similarly as long as the cost of increase is less than assigning of a new vehicle. The increase in the capacity or introduction of new tracks continues until all customers are serviced. The ray can be rotated more than once until all the customers are assigned to vehicles.

Ochi et al. (1998a), used a different measure that penalizes the unused capacity by the cost of the vehicle, aiming to minimize the cost times the unused capacity. Lima et al. (2004) report promising results using this method. The vehicle to be used is selected to have the least value of $(Q_{[i]} - Q(S)) * f_i$ among all vehicles. The vehicle selected is used as long as there is free space. The algorithm then selects back the smallest capacity and increases the capacity as long as the capacity is to be exceeded with the inclusion of a new customer.

The resulting clusters are assigned are routed using a Genetic Algorithm proposed by Önder (2007) which uses conventional heuristics as crossover and mutation operators. The CONCORDE (Cook, 2007) was not used for routing purposes as the program needs at least 16 cities to perform the routing. The clusters generated by the sweep algorithm contain less than 15 customers in more than 80% of the instances. Inter-cluster movements are not allowed at this stage, the optimum routes are found for each cluster.

4.3 Genetic Algorithm for HVRP

The genetic algorithm for HVRP is based on the study conducted by Demir (2004). The study implemented a genetic algorithm that uses conventional heuristics as operators. Constrained VRP problems, VRPPD and VRPB, were successfully solved by Demir (2004). The author used five different applications on these constrained problems and concluded that repairing infeasible tours gives the best results for constrained problems. The author used a crossover that preserves the edges in the parents, which is based on Nearest Neighbor Heuristic. The operator, which is named as the Nearest Neighbor Crossover (NNX), constructs a union graph of parental edges and the nearest neighbor is applied to this union graph. The pseudo code for the GA we have applied can be seen in Figure 4.2.

```

WHILE NOT all cutomers assigned DO
  FOR vehicle v = 1 to J
    Vehicle type  $v_t = 1$ 
    FOR  $\theta = 0$  UNTIL  $\theta = 2\pi$ 
      IF Customeri( $\theta$ ) is equal to  $\theta$ 
        AND IF distance(Customeri, center)  $\leq$  radius
          IF  $Q(S) + q(\text{Customer}_i) \leq Q_t$ 
            go to Assign
          ELSE IF  $f_{t+1} \geq f_t + f_m$  such that  $Q_m \geq q(\text{Customer}_i)$ 
             $v = v + 1$ 
             $v_t = 1$ 
            go to Assign
          ELSE  $v_t = v_{t+1}$ 
            go to Assign
        END
      (Assign) Assign Customeri to vehicle v
      FOR all customer assigned to vehicle v
        sum = cutomer(x,y)*q(customer)
      END
      center = sum / (number of customers)
      FOR all customers assigned to vehicle v
        IF distance (customer, center) > distmax
          Update  $\sigma$ 
        END
      Update  $\rho$ 
    END
  END
END

```

Figure 4. 1 Pseudo code for sweep algorithm for HVRP

Parameters and initial conditions of our genetic algorithm are based on the detailed analysis on NNX conducted by Önder (2007). The author reports that the NNX gives best solutions on TSP instances when the initial population is generated randomly. Moreover, it is reported that better results are obtained when the worst parents are replaced by their children. The algorithm initiates with the vehicle having the least capacity, and the capacity is increased using the method proposed.

The initial population is either randomly generated or use tours generated by the sweep algorithm for the initialization of the genetic algorithm. The procedure begins with selection of a vehicle with the least possible capacity, and then randomly selected customers are assigned to this vehicle as long as the capacity of the vehicle is not exceeded.

```

BEGIN
Generate the initial population (N)
  IF NOT all customers assigned DO
    Assign vehicle capacity to be  $Q_{[1]}$ 
    DO until NEW VEHICLE
      Randomly select a customer
      Calculate the load including the customer
      If the city to be visited exceeds
        Find minimum  $Q_i - Q(S) * f_i$ 
        IF minimum  $Q_i$  equal to current  $Q_i$ 
          NEW VEHICLE
        ELSE Update the capacity to be  $Q_i$ .
      END
    END
  END
END
Do until convergence observed:
  Select k parents at random from this population
  Generate a union graph from k parents
  All the edges from parents are listed in the union graph
(new) Assign capacity  $Q_{[1]}$  to the vehicle
Start from a random customer
Calculate the load including the customer
IF the city to be visited exceeds
  Find minimum  $Q_i - Q(S) * f_i$ 
  IF minimum  $Q_i$  equal to current  $Q_i$ 
    Vehicle is full
    Go to new
  ELSE Update the capacity to be  $Q_i$ .
Select the customer nearest to the current form union graph
IF the next customer is the depot
  Go to new
Replace half of the parents with the best half of the children
END
Report best solution found

```

Figure 4. 2 Pseudo code for GA algorithm for HVRP

When the insertion of a randomly selected customer exceeds the vehicle capacity, one of the tree different methods are used to select the capacity if it is necessary to increase. Figure 4.2 demonstrates the pseudo-code for the case where the Ochi method is used to select a better capacity. The new capacity with minimum Ochi value is selected, if the Ochi value of the current capacity is the best, then the vehicle is assumed to be full. This process is repeated until the vehicle is full, and then a new vehicle with least capacity is selected. This procedure is repeated until all customers assigned to arbitrary vehicles. Note that similarly the method of Reunad

and Boctor (2002) or random change is applied, one of the update methods is chosen randomly.

The assignment of all customers to vehicles determines an individual. The initial population is created when the number of individuals generated equals to number of customers in the problem. Each individual is represented with the number of the cities visited. The depot is represented by a 0. The fitness function is easy to calculate and subtours can easily be identified. The length of the tour is multiplied by the weight factor specified in the benchmark problem instances added to the cost of vehicles. This total cost is assumed as the fitness of the individual. The individuals with a small fitness value replace the parents with larger fitness value. A sample representation can be the following.

Individual_i: 0 – 2 – 4 – 5 – 6 – 0 – 1 – 3 – 7 – 8 – 0

This individual represents two trucks, the first one visit nodes 2,4,5,6 in the given order, and the second one visits 1, 3, 7, 8 in the given order. The vehicle types used by this individual are the vehicles with least capacity that can accommodate each of these two tours.

One iteration of the algorithm, between the selection and replacement steps is called as one generation. A generation of a GA using NNX is explained in detail below.

There are different approaches in the literature for selection of the parents. The individuals with best fitness values are given a greater chance of being selected, or are selected randomly (Larrañaga et al., 1999). Önder (2007) demonstrated that NNX gives best results then the parents are selected randomly, he argues that favoring the better individuals with NNX causes premature convergence on TSP problems. Thus, the parents are selected randomly. NNX makes it possible to construct the union graph using edges of more than two parents. Different parent numbers are experimented to find out the best parent number.

The construction of union graph can be demonstrated by a simple example, assume that only two parents are used to construct the union graph on a simple problem instance with 8 cities. The parents selected for reproduction are:

Parent 1: 0 – 2 – 4 – 5 – 6 – 0 – 1 – 3 – 7 – 8 – 0

Parent 2: 0 – 1 – 4 – 7 – 2 – 5 – 0 – 3 – 6 – 8 – 0

Both parents represent tours with two vehicles. Then the corresponding union graph is given in Figure 4.3, each customer is listed with the neighboring edges in increasing distance from the current customer:

0	→	1 (18)	–	5 (26)	–	2 (32)	–	6 (40)	–	8 (50)
1	→	0 (18)	–	3 (25)	–	4 (15)				
2	→	4 (10)	–	0 (32)	–	7 (28)	–	5 (35)		
3	→	1 (25)	–	7 (30)	–	8 (45)	–	6 (47)		
4	→	2 (10)	–	1 (15)	–	5 (30)	–	7 (36)		
5	→	6 (12)	–	0 (26)	–	4 (30)	–	2 (35)		
6	→	5 (12)	–	8 (35)	–	0 (40)	–	3 (47)		
7	→	2 (28)	–	3 (30)	–	8 (30)	–	4 (36)		
8	→	7 (30)	–	6 (35)	–	0 (50)				

Figure 4.3 A sample union graph for NNX

The numbers in the parenthesis correspond to the distances between two cities. The individual that will be generated using the NNX is going to start with 0 the depot. Then the next individual to be selected is 1, which is 18 units far from the depot. Then after 1, 0 is selected. The first vehicle will visit 0 – 1 – 0. Then 5 will be visited as it is nearest to the depot. 5 yields to 6, and 6 yields to 8. 7 will be visited after 8. Note that the Ochi values are checked at each iteration, assume that the Ochi value of including 7, is not better than the Ochi value when 7 is considered, then the second vehicle will visit 0 – 5 – 6 – 8 – 0. The algorithm will select 2 as it is nearest feasible neighbor in the parents. Then 4 will be selected, and 4 has no feasible neighbor. The algorithm gets stuck, the nearest among all possible customers, which can be included in the vehicle regarding the Ochi value, is selected. 3 is included if the distance is less than 7, which are the remaining unvisited customers. 3 yields 7 according to the union graph. The last vehicle in the child generated from this union graph will visit 0 – 2 – 4 – 3 – 7 – 0. The child generated is then:

Child: 0 – 1 – 0 – 5 – 6 – 8 – 0 – 2 – 3 – 7 – 0

Note that the edges, between 0 – 1, 5 – 6, 8 – 0, 0 – 2, 3 – 7 from the first parent are preserved, and edges 0 – 1, 0 – 5, 6 – 8, 8 – 0, from the second parent are preserved, while, the edge 3 – 7 is introduced in the new individual. NNX has the ability of discovering new edges while aiming to preserve good parental edges. This child replaces one of its parents if the total cost of visiting the cities in these orders with the least cost vehicle alternatives is less than the fitness of its parents.

The newly generated individuals are subjected to node insertion mutation that improves the results of NNX according to Önder (2007). A randomly selected customer is removed from the assigned vehicle. All edges are checked and the removed customer is tried to be inserted to an improving point. A node insertion move is completed if a feasible insertion point for the removed customer is found. This mutation operator is tried for 15 times as proposed by Önder (2007) on each child that is better than its parents. It must be mentioned that this mutation is relatively simple as only one node can be removed and reinserted, Renaud and Boctor (2002) proposed an eleven step improvement using 2-opt and 3-opt moves.

The individual with least cost is reported after the algorithm has converged. The convergence can be measured using different methods. Having population average equal to the best individual is generally used as a convergence measure, on the other hand, most of the combinatorial optimization applications using GA have a generation limit as the convergence criterion, as it takes very long time for population average to converge to the population best.

4.4 Multi-Agent Solution for HVRP

The study is the first application of multi agent systems to HVRP. Multi-agent systems are gaining importance in the realm of Artificial Intelligence (AI) applications, where more than one decision maker is involved or the problem has a distributed nature. Multi-agent systems consist of autonomous agents that interact with each other in order to reach pre-specified solution (Erdođdu (2004)). Autonomous agents are used to model rational entities, like independent decision makers or entities that work for the same goal but are physically separated.

There is a few works on direct solution applications of the well-know VRP instances to the knowledge of the authors. However, multi-agents approach has been widely used in distributed planning. Erdođdu (2004) summarizes the most significant work by stating that Boutilier and Brafman (1997) worked handling multi-agents that share the same goal and execute some action concurrently, Bowling et al. (2002) introduced some game theoretic approaches and Ephrati and Rosenschein (1997) used robust planners for generating dynamic plans.

Fischer et al. (1999) describes a comprehensive simulation of a truck distribution environment with more than one company and more than one truck, which can be referred as a multi depot VRP. A trade mechanism, which is quite similar to the one we implemented is used. Firstly, price for the exchange of a city is calculated, one agent calculates the saving when it removes a specific city from its solution, while the other calculates its loss it incorporates when that specific city is added to its current solution. This procedure is overseen by another agent that keeps track of data, and manages the transactions. The result is improvement in the current schedule if there is a transaction.

Thangiah et al.'s (2001) work is conceptually similar to Fischer et al. (1999) again there is a centralized agent that at the beginning of the solution delivers the cities to agents that model the vehicles. A bidding mechanism is constructed in order to find out the most suitable vehicle for a city, every vehicle announces its savings by accepting a new arrival, and then the best bid is assigned with the city. The most significant difference of this work is an improvement mechanism is used, where the vehicles exchange the cities between each other, directly based again on the savings relative to each other.

In their recent work, Boudali et al. (2004) concentrates on the VRP with Time Windows, the cities must be served within a given time interval, and models every city as an agent. This is a totally different approach. Coalition formation is done by interaction among agents, where every agent asks its non-enemy agents, if they want to coalesce with itself, and they forward that message to their ancestors. When an agent finds out that it can form a coalition with the current one it sends back a message, accepting this proposal. When a coalition is formed and no other agent can

enter the solution, the agents announce a friendship and other declare them as enemies, thus a new coalition shall be formed. By this way forming of new coalitions, with autonomous agents that try to maximize their own benefits form the solution by negotiating between each other.

Lastly, Erdoğan (2004) concentrates on resource based planning, re-planning of Dynamic Postmen Problem. The problem is similar to VRP, where new cities arrive, and the postmen (vehicles) are the agents. The agents have an initial list of nodes to be visited, and after a plan is generated a new node arrives, the model is based on the negotiation for the assignment of the new node after the plan has been formed. The negotiation is resource based, which is similar to the negotiation mechanism discussed.

MA applications are only limited to VRP applications, and no study aimed to solve HVRP. In our implementation, we use two types of agents to handle the HVRP. The first type is the vehicle agents (VA) and the second is the central agent (CA). All the agents are autonomous reactive agents; they react to the changes and act rationally, without reasoning about the changes. The tasks each agent will perform are described including the necessary structures.

Vehicle Agents

The vehicle agents have three main tasks; the first one is to keep the route that traverses all the cities, namely Hamiltonian Cycle, as short as possible, the second one is advertising cities to other agents, and the very last one is giving bids for the advertised cities.

Each agent is initialized using by a randomly assigned city. The agents are capable of changing the capacities of vehicles assigned, including the price of updates to the cost calculations.

An agent keeps an ordered list of cities to be visited; the order of the list represents the order in which the agent is going to visit the cities. Thus, the agent should optimize the order in order to minimize the route, this is done by the help of the 2-opt heuristic.

The next task the vehicle agents serves is finding the city that results in maximal shortening in the tour. This is similar to the first part of the savings equation. Let us assume that (for agent t) city i is connected to city j , and city j is connected to city k , (the visiting order of the agent is $\dots - v_i - v_j - v_k \dots$). Then shortening in the tour can be calculated by: $s_{jik} = (c_{ij} + c_{jk}) - c_{ik}$ (as the new visiting order will become $\dots v_i - v_k \dots$ remaining j to be advertised). The city with maximum savings is obtained by:

$$s_j = \max \{ s_{jik} \mid \forall i, j, k > 0 \text{ and } i \neq j \}$$

$$j = \operatorname{argmax} \{ s_{jik} \mid \forall i, j, k > 0 \text{ and } i \neq j \},$$

the city j and the savings are announced to a common blackboard.

The blackboard is used as the communication medium; all the agents can see the values on the blackboard and can decide on their actions accordingly.

A modified version of the savings announcement is the “exaggerated” announcement implemented as an alternative. In this version when an agent observes that the capacity usage ratio of it is less than the average of all other vehicles, and the city it advertises keeps remaining on the blackboard for consecutive iterations, it overstates its savings, by the following formulae:

$$s_j^x = s_j \ln(1 + n \times U(c))$$

where n denotes the number of instances the agent advertised city j consecutively, $U(c)$ denotes the capacity utilization of the agent. The equation is logarithmic to ensure that the increase of the s_j^x value is decreasing avoiding illogical (very high) values to be advertised. Moreover, the $U(c)$ takes part in the equation to ensure that the exaggeration is inversely proportional to the usage rate, while increase proportionally by the number of instances the agent advertised the same city consecutively.

The vehicle agents lastly calculate the total saving of inserting the cities they see on the blackboard by using an equation similar to the one discussed above. Assume that an agent (agent u) whose has the arc $\dots v_x - v_y \dots$ on its current route, sees the following information on the blackboard, then it calculates its own bid value.

Blackboard: “ j ” $\rightarrow s_j$

Agent u calculates: $S = (c_{xj} + c_{jy}) - c_{xy} + s_k$

The savings S being larger than 0 and agent having enough capacity to accommodate the demand of j , the agent announces its bid to be S and the place (v_x, v_y) to the central agent.

Again there is one alternative to the calculation of the bid S (exaggeration version), if the agent an agent observes that the capacity usage ratio of it is less than the average of all other agents, S is decreased using the following equation:

$$S^x = S \times U(c) r/10$$

where r denotes the number of bids the agent has offered since the beginning of the run. As the value $U(c) \leq 1$, then S^x decreases with the increase in r . In this study $r/10$ is used as the instances algorithm usually terminates a bid number which is less than 10 times the number of nodes, and 10 ensures that the S^x value does not approach 0 during a run.

This “exaggeration” structure was employed to avoid a city continuously being traded from one agent to other and vice versa. The standard version of the algorithm was observed to result in infinite loops where the same city that is taken from an agent in one term is assigned back to the same agent at the next term. Yet when an algorithm is based on capacity usage ratio then if a city is taken from one agent that agent will tend to bid a lesser amount as the capacity usage ratio would change.

Central Agent

The central agent acts as the final decision maker, for the bidding mechanism, its function is the same as the manager agent in Fisher et al. (1999). The agent decides on the exchange of cities between agents after all the agents announce their bids. The decision mechanism has two different versions implemented; the first one is deterministic version. In this version the central agent select the best offer for each city and assigns the city to the position announced by the agent that gave the bid, deleting it from its current position on the advertising agent. It shall be mentioned

that at each iteration step central agent makes only one change on the vehicle agents. Thus if an agent gives a city, it cannot get a city until the next iteration.

A modified version of the central agent is also present; in this version, the central agent selects the city to be assigned is selected proportional to the probability that is calculated using the following equation:

$$p_i = \frac{1}{\eta} \left[\eta - 2(\eta - 1) \left(\frac{i-1}{m-1} \right) \right]$$

The index i denote the rank of the agent among the m agents who give bids for a city advertised. The parameter η denotes the pressure to select the good bid. η varies between 1 and 2, 1 denotes the totally random selection as all the positive bid giving agents have equal probability of winning the bid. 2 denotes the alternative where the best bid has the most chance of being selected.

The trading continues until a feasible bid is observed. The vehicle type of each agent and the resulting tours are the output of this solution approach.

CHAPTER 5

NUMERICAL EXPERIMENTS AND COMPARISON OF THE RESULTS OF IMPLEMENTED ALGORITHMS

The algorithms discussed in Chapter 4 are implemented with necessary modification to find one best way of handling the HVRP and HFFVRP problem. The results of the algorithms are reported as percent deviations from the LP lower bounds calculated in Chapter 3. The percent deviations from the best known solution is also reported for comparison reasons for the solutions that use the objective function that aims to minimize the cost of vehicles and cost of routing simultaneously.

The CFRS method is investigated in detail in this chapter as the genetic algorithm and multi-agent algorithm uses the sweep algorithm as the initial point. Different clustering methods are analyzed and compared; the chapter finishes with the comparison of the results of the algorithms.

5.1 Cluster-First-Route-Second Algorithm

Various parameter levels and methods are implemented to obtain the best parameter combination for our sweep algorithm.

The radius of convergence uses the multiplicative inverses of capacity usage to include the nodes that in the immediate neighborhood while the capacity utilization is low, and increase the coverage of the neighborhood as the capacity utilization increases. This gradual increase of the radius of convergence (ρ), aims to include the far away customers as long as there is free capacity, and they are not really far away.

Different levels of the parameter R are used, when this shape parameter is very small, the algorithm tends to include very few customers on each route, as no other customers are within the ρ . When R is very large, ρ increases such that no limitation on the cities to be included in the route remains then the algorithm becomes pure sweep algorithm.

As mentioned in Section 4.2 three different methods are implemented when the vehicle type is to be changed to increase the capacity of the vehicle. All the methods are implemented on the problem sets defined in chapter 3. The result of the algorithm is influenced by the clustering structure of the problems, as the sweep algorithm is expected to work well with clustered instances.

Another factor that affects the results of the sweep algorithm is the initial point where the ray of infinite length is started to be rotated as the capacity usages and the vehicles selected are dependent on the order the customer are included in a vehicle. We have initiated the sweep algorithm such that each customer became the first to be included in a vehicle once. The clusters are finalized when the capacity is full or there is no customer within the radius of convergence.

The shortest route between points within a cluster and the depot are routed using the genetic algorithm proposed by Önder (2007), without allowing the inter-cluster movements. The inter-cluster movements are restricted as the quality of the clusters generated is under consideration at this stage of the CFRS algorithm. (The C code for CFRS algorithms is given in Appendix D.)The result of the best possible initial point is reported.

Six different R values are used in the experimental runs, 0.5, 1, 1.5, 2, 2.5 and 3. There are two different alternatives for calculating σ , one including the depot the other not. Furthermore, there are three vehicle selection methods, Renaud and Boctor (2002), Ochi et al. (1998a) and selecting randomly. There are 12 different problems with varying sizes in both of the sets. Analyses of results of 864 different replications are reported in Appendix A, the factor analysis and main interaction plots for our model are given in Figure 5.1 and 5.2. The appendix also includes the

analysis where the problem is solved as a pure HVVRP using the same parameter settings. MINITAB for Windows13.3 is used to conduct the factor analysis.

The main interaction plot is used for analyzing the affect of changes in discrete levels of parameters. The affect on the deviation from the LB of three decision parameters can be seen in Figure 5.1. However, the factor analysis shall be conducted to analyze the statistical significance of changes in parameters. The results of the ANOVA analysis are used to comment on the parameters. For our model, the residuals of the model can be assumed normal, as the p-value of Anderson-Darling normality test are 0. However, the histogram and normality plot tests indicate that the residuals are not distributed normally. The size in the analysis is kept as a block in the factor analysis, the results of the

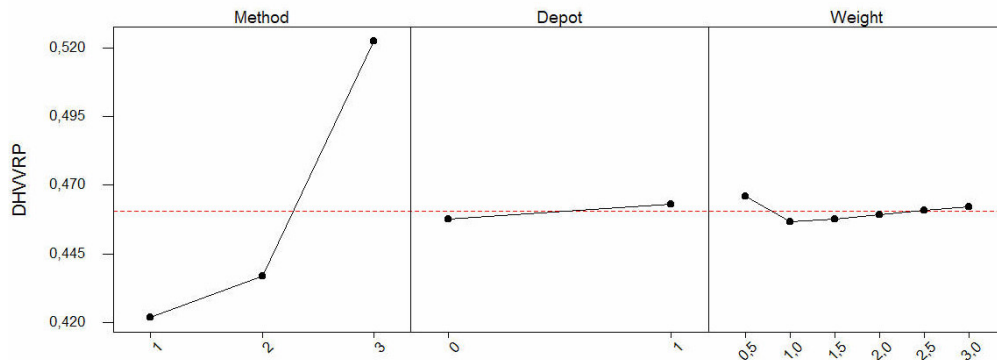


Figure 5. 1 Main effect plots for the deviation from the LB for our model

According to the ANOVA results the method used for selection of larger vehicles, is statistically significant. It can be seen that R (Weight) value is not significant statistically, thus the weight is not further considered. The presence of the depot in the solution is also not statistically significant.

According to figure 5.2 the weight and the inclusion of the depot into the calculations do not have an impact on the solutions, when the interaction is considered. There is no complicated interaction among the decision parameters.

Another factor analysis we have conducted, is the inclusion of the clustered structure of the problem. The clustered structure is not a decision parameter for the problems solver, yet the interaction of this parameter, if statistically significant, can play an important role in selection of the best course of action.

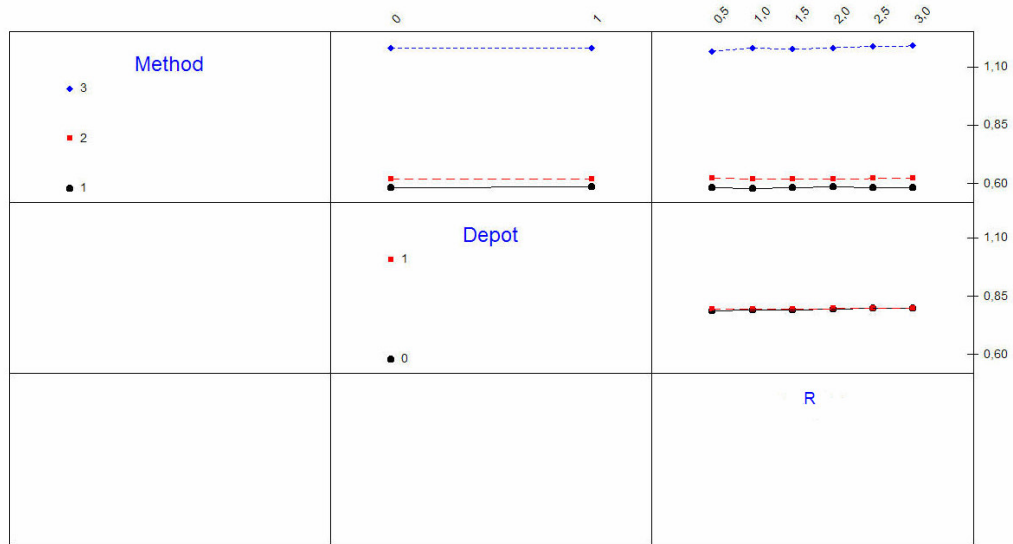


Figure 5. 2 Interaction plots for the deviation from the LB for our model

The Figures 5.3 and 5.4 are the main effects plots when the clustered structure is included in the analysis. The ANOVA and the normality plots of the residuals are given in Appendix A. Figure 5.3 demonstrates the effects when the model we have developed is used, and the corresponding LB are considered when the deviations are calculated. Figure 5.4 demonstrates the effects when the problem is solved as a HVRP, to minimize the total cost. The figures are similar in when the capacity increasing method and weather to include the depot into the cluster or not, are considered. The change in the R can be explained by the difference in the tightness of the LB for our model and the HVRP. The types of vehicles available is a parameter that can be adjusted, thus the analysis can be helpful in developing the problem structure. The type of vehicles demonstrates a great difference in the graphs as the costs of vehicles have a great impact on the objective values of both of the problems.

There is a significant difference, when the clustering is considered. The sweep algorithm we have employed, gives better results when the problems are not clustered when our objective function. On the other hand, when the distances are included in the cost is used in HVRP, the sweep algorithm gives better results when the problem is clustered.

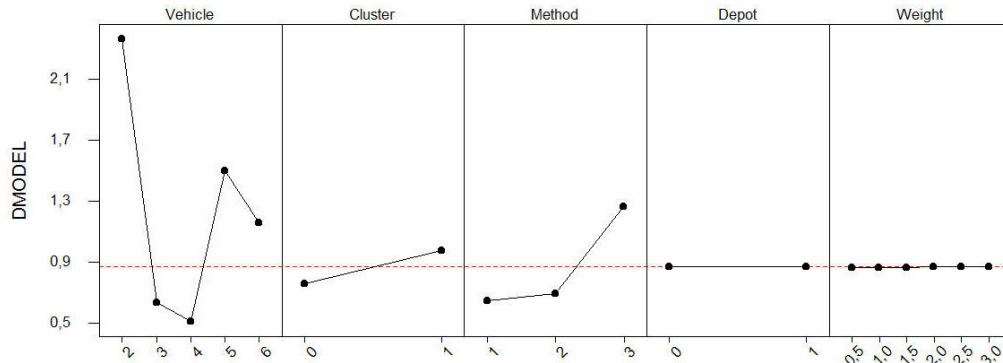


Figure 5.3 Main effect plots including “cluster” for the deviation from the LB for our model

The average deviation from the LB is given in Table 5.1. The deviations are extremely high. The results of Ochi and Boctor are similar, while the Random selection gives the worst results. The deviation is less when the problem size increases, this is possibly due to the improvement of the lower bound.

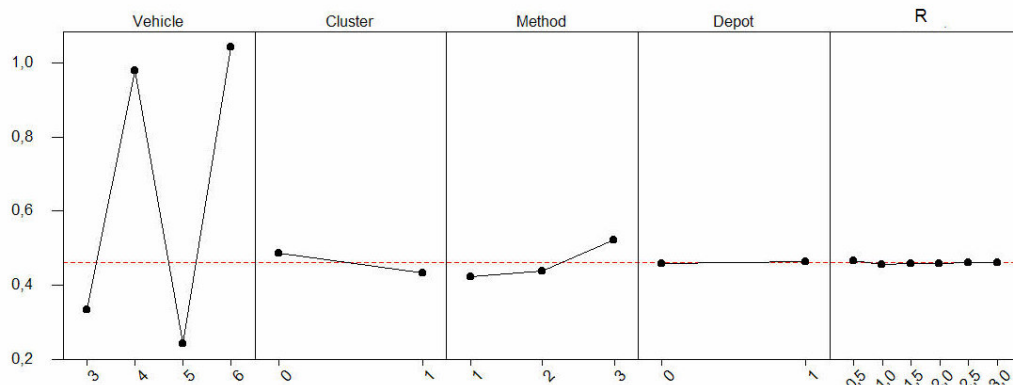


Figure 5.4 Main effect plots including “cluster” for the deviation from the LB for HVRP

As the LB generated for our model is not tight, a comparison with the best known solutions can be handled better when the HVVRP is considered. The deviation from the best-known solutions of the Golden et al. (1984) instances is given in Table 5.2. The best results that can be obtained using the mentioned method with the given parameters are listed in the columns according to the problem number. The results for problem 17 and 18, which contain a 5 and 6 vehicles respectively, are very poor.

Table 5. 1 Average percent deviation from the LB for sweep algorithm on our model

Method	Size				Average
	20	50	75	100	
Boctor	96.66	50.03	44.86	11.70	58.32
Ochi	100.80	57.41	44.86	11.70	62.16
Random	125.63	95.58	207.06	58.58	118.01
Average	107.70	67.67	98.93	27.33	98.82

The results reported by Renaud and Boctor (2002) are better compared to the results presented in Table 5.2 This is due to the post improvement phase they have used after the sweep algorithm. In our implementation the clustering ability of the heuristics is analyzed, and the resulting clusters are routed in the best possible route. However the improvement moves Renaud and Boctor (2002) used contain inter-cluster moves, 2-opt moves, and edge exchanges. The results shall be improved by improvement heuristics if only sweep algorithm is to be considered. The deviation from the best known solution is less than 20% for 8 out of 12 problem instances.

5.2 Genetic Algorithm

The genetic algorithm described in Section 4.3 is implemented with different parameter settings. (The C code of the algorithm is given in Appendix D.)The algorithm is either initialized randomly or using the clusters generated by the sweep algorithm. The clusters generated by the sweep algorithm are used with the form obtained prior to routing in order to avoid the algorithm getting trapped to local optima easily. The clusters generated by the Boctor method are used as an input when the initial population is not generated randomly, according the results of the previous section.

The genetic algorithm initializes with a population that is equal to the number of customers in the problem. The algorithm is run for 10,000 generations. The parents are randomly selected and a child is generated using the nearest neighbor list. The children that are better than their parents are replace their parents after a simple mutation. The mutation consists of node exchange move that is applied for 15 times on each child. The best member of a population is reported at the end of the algorithm. 10 different replications are conducted for each problem setting.

Table 5. 2 Minimum percentage deviations from the best-known solutions of problem set 1

Method	Weight	Golden 3	Golden 4	Golden 5	Golden 6	Golden 13	Golden 14	Golden 15	Golden 16	Golden 17	Golden 18	Golden 19	Golden 20
Bactor	0.5	14.17	15.23	29.57	14.55	39.10	11.42	18.09	27.16	60.77	91.25	6.70	19.39
	1	15.14	15.22	29.61	14.66	39.42	11.43	17.77	27.73	64.84	92.48	6.22	25.50
	1.5	15.14	15.22	29.61	14.74	41.78	17.15	19.03	29.20	64.84	92.48	6.52	26.04
	2	15.14	15.22	29.61	14.74	41.78	17.15	19.16	29.20	64.84	92.48	6.65	26.09
	2.5	15.14	15.22	29.61	14.74	41.78	17.15	19.16	29.20	64.84	92.48	6.65	26.10
	3	15.14	15.22	29.61	14.74	41.78	17.15	19.16	29.20	64.84	92.48	6.65	26.10
Ochi	Min	14.17	15.22	29.57	14.55	39.10	11.42	17.77	27.16	60.77	91.25	6.22	19.39
	0.5	13.35	15.23	23.28	14.55	50.39	16.94	22.31	31.59	60.77	91.25	6.70	19.39
	1	14.33	15.22	25.36	14.66	51.26	18.22	21.55	34.64	64.84	92.48	6.22	25.50
	1.5	14.33	15.22	25.36	14.74	51.43	18.71	23.09	32.68	64.84	92.48	6.52	26.04
	2	14.33	15.22	25.36	14.74	51.43	18.85	23.42	32.70	64.84	92.48	6.65	26.09
	2.5	14.33	15.22	25.36	14.74	51.43	18.87	23.54	32.70	64.84	92.48	6.65	26.10
Random	3	14.33	15.22	25.36	14.74	51.43	18.87	23.54	32.70	64.84	92.48	6.65	26.10
	Min	13.35	15.22	23.28	14.55	50.39	16.94	21.55	31.59	60.77	91.25	6.22	19.39
	0.5	16.40	14.87	8.26	13.45	68.70	33.85	35.90	37.31	41.75	127.24	46.63	51.42
	1	9.28	21.72	7.43	20.19	57.43	34.40	40.30	30.51	46.39	124.72	42.65	50.04
	1.5	8.24	21.72	9.21	14.00	60.51	34.35	39.02	32.00	47.96	124.82	42.58	51.22
	2	8.24	21.72	9.21	14.00	60.51	34.37	39.16	32.24	48.39	124.88	43.37	53.06
Min	2.5	8.24	21.72	9.21	14.00	60.51	34.37	39.16	34.32	48.39	127.87	43.52	53.66
	3	8.24	21.72	9.21	14.00	60.51	34.37	39.16	34.32	48.39	127.94	43.83	53.91
	Min	8.24	14.87	7.43	13.45	57.43	33.85	35.90	30.51	41.75	124.72	42.58	50.04
	Min	8.24	14.87	7.43	13.45	39.10	11.42	17.77	27.16	41.75	91.25	6.22	19.39

Table 5.3 the best results of 10 replications. The type of the initial population used is given in the third column. The results regarding our model are given along with the results of the HVRP solution for easy comparison. The deviation from the best-known solution is reported for the first problem set.

Table 5. 2 Results of Genetic Algorithm on problem set 1

	Initial Pop	Size	CPU Time	Result	Dev LB (%)	CPU Time	Result	Dev LB (%)	Dev BN (%)
3	Random	20	0.00	490.00	20.57	0.60	989.55	5.06	2.97
	Boctor	20	0.00	480.00	18.11	0.10	990.07	5.12	3.02
4	Random	20	0.00	5657.28	47.29	0.00	6962.64	8.75	8.16
	Boctor	20	0.00	6500.96	69.26	0.00	7423.17	15.94	15.31
5	Random	20	0.30	573.91	47.16	0.80	1089.41	29.05	8.18
	Boctor	20	0.00	521.29	33.67	0.60	1125.79	33.36	11.79
6	Random	20	0.00	4603.56	19.86	0.10	7470.31	18.24	14.64
	Boctor	20	0.00	7091.48	84.63	0.00	7531.39	19.20	15.57
7	Random	50	1.70	1375.71	29.65	1.50	3267.33	51.54	35.78
	Boctor	50	1.00	1260.00	18.74	1.10	3283.00	52.26	36.43
8	Random	50	1.90	9533.28	48.21	1.60	10273.60	14.48	12.66
	Boctor	50	1.00	9850.13	53.13	1.00	10623.22	18.37	16.50
9	Random	50	1.60	1918.65	92.66	1.30	2936.46	33.56	13.54
	Boctor	50	1.00	1801.99	80.94	1.00	3135.09	42.59	21.22
10	Random	50	1.90	1975.41	98.36	2.30	3514.07	37.45	29.17
	Boctor	50	1.00	2236.02	98.36	1.00	3791.69	48.31	39.38
11	Random	75	1.70	1039.70	48.30	2.20	2254.80	53.52	29.23
	Boctor	75	2.00	889.40	48.30	1.90	2402.11	63.55	37.67
12	Random	75	1.60	1147.85	48.11	1.60	3811.86	74.92	60.74
	Boctor	75	2.00	1557.27	48.11	1.90	3879.99	78.05	63.61
13	Random	100	7.00	8860.88	21.38	7.60	11185.81	37.87	29.10
	Boctor	100	4.00	8143.63	21.38	4.00	9606.68	18.41	10.88
14	Random	100	10.60	3402.72	36.11	10.90	5247.20	47.97	29.90
	Boctor	100	4.00	3014.18	20.57	5	5194.1044	46.47	28.58

For the problem set 1, the average deviation from the best known solutions for the problem set 1 is 21.21%. While the deviation from the LB for our model is 41%, and for HVRP model is 31.66. The average deviation from the LB for our model is 26.34%,and for HVRP model is 19.56.

The improvement the GA brings in on the solutions generated using the Boctor method is given in Table 5.5. The difference is calculated by simple subtracting the average percent deviations. The results suggest that the improvement is 14.39%. More problem specific heuristics shall be incorporated to intensify the search.

Table 5. 3 Results of Genetic Algorithm on problem set 2

	Initial Pop	Size	CPU Time	Result	Dev LB (%)	CPU Time	Result	Dev LB (%)
1	Random	20	0.00	390.00	32.56	0.40	920.36	14.57
	File	20	0.00	490.00	28.58	0.10	940.47	17.07
2	Random	20	0.00	4012.37	4.46	0.00	6928.15	11.58
	File	20	0.00	6500.00	69.23	0.00	6880.09	10.80
3	Random	20	0.00	267.34	30.41	0.30	920.36	2.36
	File	20	0.00	490.00	139.02	0.10	940.47	4.59
4	Random	20	0.00	5023.49	30.79	0.10	6928.15	10.50
	File	20	0.00	6500.00	69.23	0.00	6880.09	9.74
5	Random	50	1.80	1310.04	19.09	2.00	3116.44	27.94
	File	50	1.20	1263.60	14.87	1.00	3426.16	40.65
6	Random	50	1.10	7669.99	19.24	1.00	9550.76	7.97
	File	50	1.00	8571.79	33.26	0.90	10048.13	13.60
7	Random	50	2.20	1417.81	42.37	2.60	3006.87	23.04
	File	50	1.10	1762.07	76.93	1.10	3012.15	23.26
8	Random	50	2.60	1905.66	91.35	2.70	3380.70	34.11
	File	50	1.70	2206.70	91.35	2.00	3336.61	32.36
9	Random	75	2.00	1014.16	44.65	2.10	2046.64	55.90
	File	75	2.00	832.32	44.65	2.20	2126.45	61.98
10	Random	75	1.50	1460.97	77.93	1.00	3352.97	59.94
	File	75	2.90	1162.96	77.93	2.00	3891.34	85.62
11	Random	100	6.00	7804.63	6.91	4.50	9906.50	23.99
	File	100	2.00	8119.12	6.91	2.00	9569.28	19.77
12	Random	100	8.60	2902.14	7.49	9.80	4890.22	27.95
	File	100	3.40	2784.19	3.12	3	4797.9291	25.54

5.3 Multi-Agent Solution for HVRP

The agent system described in Section 4.5 is implemented on both of the problem sets, different parameter settings are used to improve the results. (The C code of the algorithm is given in Appendix E). The behavior of agents is controlled using the selection and exaggeration parameters. According to previous experiments the agents tends to get stuck in local optima when they announce the real savings they gain, when an exchange is done. Golden et al. (1984) demonstrated that modifying the costs of gain give better results, compared to the pure savings algorithm on HVRP. Thus all the agents exaggerate the savings when they are announcing, or understate the gain when they obtain a city.

Table 5. 4 Average improvement of deviation from the LB with GA

	Size				
	20	50	75	100	Average
Boctor (Dev %)	96.66	50.03	44.86	11.70	50.81
GA on Boctor (Dev %)	52.85	48.27	43.54	5.02	36.42
Improvement (Dev %)	43.81	1.76	1.32	6.69	14.39

The selection done by the central agent is done probabilistically, controlled by the parameter η . Selecting always the best bid, sometimes causes the algorithm to get stuck at local optima. 5 different levels of η are used in the experiments as small changes in η , cause changes in final solution. As the algorithm is probabilistic, 10 different replications of each setting are implemented.

The agents are either honest or they exaggerate the results they are gained if the city is transferred from them. Similarly the agents tend to exaggerate the saving they obtain when they are acquiring a city.

Table 5.6 and Table 5.7 summarize the results for different levels of η (Etha). The results are very poor compared to the results in the literature. The results are better than the results of the pure savings algorithm for small problems when compared, yet they are not good enough to be applied under these settings.

Table 5. 5 Percentage deviations of different η values for MA, using our model

Problem	Etha					Minimum
	1	1.25	1.5	1.75	2	
Golden 3	25.55	24.86	22.43	26.70	13.65	13.65
Golden 4	25.75	33.18	17.84	26.42	24.65	17.84
Golden 5	56.71	47.35	52.35	45.21	36.14	36.14
Golden 6	37.75	22.25	21.16	36.86	19.84	19.84
Golden 13	118.14	128.94	122.99	130.49	121.50	118.14
Golden 14	55.25	53.34	43.87	59.84	50.59	43.87
Golden 15	104.90	100.45	83.80	86.11	77.11	77.11
Golden 16	88.03	96.53	89.27	86.48	63.95	63.95
Golden 17	173.51	167.60	149.53	144.36	93.40	93.40
Golden 18	162.73	171.44	165.74	156.71	122.89	122.89
Golden 19	80.03	73.25	67.35	65.25	47.14	47.14
Golden 20	123.95	125.67	123.02	114.94	75.94	75.94
Solomon 1	70.56	43.01	61.37	61.37	46.94	43.01
Solomon 2	95.27	108.28	82.31	95.27	95.29	82.31
Solomon 3	217.07	165.85	200.00	200.00	173.17	165.85
Solomon 4	95.27	108.28	82.31	95.27	95.29	82.31
Solomon 5	141.66	154.62	155.76	151.14	129.15	129.15
Solomon 6	104.55	96.33	96.30	96.91	53.71	53.71
Solomon 7	252.95	222.05	221.22	203.39	185.82	185.82
Solomon 8	193.43	197.51	186.20	174.54	144.87	144.87
Solomon 9	263.46	265.86	252.83	241.93	150.88	150.88
Solomon 10	291.51	273.49	291.35	258.13	164.54	164.54
Solomon 11	95.34	92.57	79.79	81.06	58.28	58.28
Solomon 12	146.78	134.62	127.60	115.27	83.75	83.75

The multi-agent systems fail to generate good enough solutions in our very simple implementation that is based on the saving heuristic although a 2-opt improvement mechanism is incorporated in the algorithm. The average of deviations from the LB's is more than 100% for our model, and about 59% for the HVRP.

This study is the first attempt to use agent systems in HVRP solution. The algorithm lacks some problem specific algorithms, like the γ -weighted savings algorithm proposed by Golden et al. (1984). Moreover, the 2-opt improvement method is limited to intra-cluster movement, limiting the search space. The study can be further developed using more effective heuristics in the decision mechanisms of the vehicle agents.

Table 5. 6 Percentage deviations of different η values for MA, on HVRP instances

Problem	1	1.25	1.5	1.75	2	Minimum
Golden 3	48.87	23.03	40.32	50.26	53.19	23.03
Golden 4	96.30	110.39	82.71	95.75	95.27	82.71
Golden 5	89.24	70.35	75.20	66.80	70.78	66.80
Golden 6	110.56	85.06	84.47	109.91	84.67	84.47
Golden 13	155.19	164.45	158.06	164.60	159.76	155.19
Golden 14	107.08	104.73	91.92	113.44	101.48	91.92
Golden 15	235.24	230.89	199.51	199.58	189.28	189.28
Golden 16	71.17	80.51	72.40	71.06	52.08	52.08
Golden 17	270.58	254.53	234.89	230.13	169.03	169.03
Golden 18	287.05	291.16	291.78	284.20	228.42	228.42
Golden 19	83.86	77.66	71.67	68.95	50.55	50.55
Golden 20	133.34	138.25	131.45	124.01	81.88	81.88
Solomon 1	230.9373	177.4821	213.106	213.106	185.1075	177.4821
Solomon 2	95.27	108.28	82.31	95.27	95.29	82.31
Solomon 3	217.07	165.85	200.00	200.00	173.17	165.85
Solomon 4	95.27	108.28	82.31	95.27	95.29	82.31
Solomon 5	141.66	154.62	155.76	151.14	129.15	129.15
Solomon 6	104.55	96.33	96.30	96.91	53.71	53.71
Solomon 7	252.95	222.05	221.22	203.39	185.82	185.82
Solomon 8	193.43	197.51	186.20	174.54	144.87	144.87
Solomon 9	263.46	265.86	252.83	241.93	150.88	150.88
Solomon 10	291.51	273.49	291.35	258.13	164.54	164.54
Solomon 11	95.34	92.57	79.79	81.06	58.28	58.28
Solomon 12	146.78	134.62	127.60	115.27	83.75	83.75

5.4 Analysis of the Results of the Experimental Runs

The percent deviations from the LB are very high for all of the heuristics we have experimented with in this chapter. This is possibly due to the lack of tightness of the LB generated in Section 3. The deviation for the best known solutions using the

HVRP model suggest that the deviation from the best known solutions is 29.81% on average when the CFRS method is used and 21.21% when the genetic algorithm is used.

The problems defined by Golden et al. (1984) have been widely studied and the average utilization of the vehicles is more than 90% for the best-known solutions, thus algorithms like the ones we have implemented naturally fall behind these utilization values as well. Complex moves of series adjacent cities is be required to intensify the search mechanisms of the GA and CFRS methods.

The results of the mutli-agent attempt on the HVRP and our problem has an advantage of being implemented on distributed environments like the palm computers of the drivers, however the city trade mechanisms need to be improved. A very simple implementation suggests that the agents are capable of finding solution with deviation around 20% from the LB for some instances. On the other hand on the average the algorithm does not find good solutions. A detailed analysis on the performance of the algorithm based on the topology and the cost structure can reveal useful information for better implementations of distributed artificial intelligence.

CHAPTER 6

AN APPLICATION OF THE PROPOSED ALGORITHMS

A hypothetical firm that faced the problem we have explained in Chapter 1 is analyzed in this section.

The problem of assigning of the bus-stops to the routes and routing are to considered, in two different settings. The clustering methods and R values are implemented in this chapter, and the GA is implemented both with a random initial population and the results of the Renaud and Boctor (2002) method.

The size of the problem is large, with 92 bus stops among the city. GIS software can be employed to generate the distance matrix for a real life application.

A distribution generated randomly that follows the topology of a real distribution of the bus stops is generated to analyze the performance of the solution methods proposed. Figure 6.1 demonstrates the distribution of the bus-stops (circles representing the bus-stops, squares the plant). The plant is located in a suburb of the city. The problem is not cannot be assumed to be clustered, as the borders of the suburbs of the city are not very visible on the plot. There are some stops on the road that connects the city to the plant.

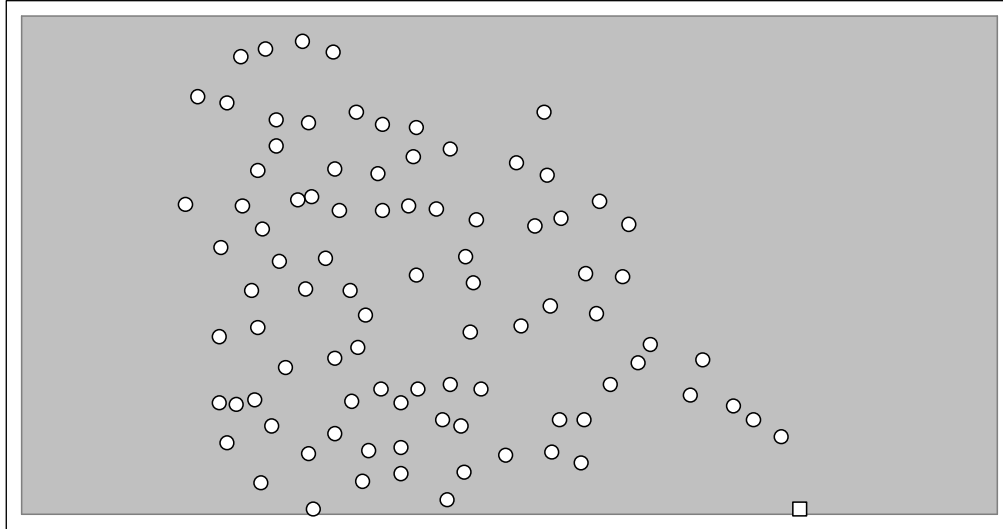


Figure 6. 1 The plot of the bus-stops and production plant of the hypothetical firm

The firm uses two types of busses with capacities of 20 and 40, respectively and the fixed cost of the large bus is two times the small bus.

The bus that leaves the plant is expected to be back at the firm in 75 minutes. The speed of the busses is assumed to be 50 km/h for the whole city. No time is allocated for passenger pickups, as there is relatively low number of passengers on each bus stop. The time to complete the tour is thus converted to km's to be 62.5 km's. The distance of the farthest point to the firm is about 20 km's in our graph.

The number of passengers is about 3-5 on average for the bus stops that are not in the city center, while this becomes around 8-10 in the city center. The bus that passes through a bus-stop is to pick-up all the passengers waiting on a stop.

6.1 Results on a Random Problem

The results on the problem generated for all the solution approaches is given in the following table. The LB is calculated using the mathematical model described in the second chapter. The LB calculated is 48000, for the problem. The results for the sweep methods is given in table. All of the methods give the same minimum cost as 55163.13, when $R = 3$.

Table 6. 1 Results of the CFRS algorithm for fort he hypothetical firm

	0.5	1	1.5	2	2.5	3
Boctor	54074.83	54191.37	54214.39	54214.39	54214.39	54214.39
Boctor With Depot	54192.23	54214.39	54214.39	54214.39	54214.39	54214.39
Ochi With	54074.83	54191.37	54214.39	54214.39	54214.39	54214.39
Ochi With Depot	54192.23	54214.39	54214.39	54214.39	54214.39	54214.39
Random	50224.93	50102.21	50178.44	50178.44	50187.08	50246.78
Random With Depot	62842.93	50187.08	50246.78	50246.78	50246.78	50246.78

However, the genetic algorithm resulted in 52800.28, when it is initiated randomly, and 52615.38, when the initial population is generated using the sweep algorithm with the method proposed by Renaud and Boctor (2002), for the minimum of 10 replications is considered. The clustering when the GA is run initialized with the sweep algorithm is done by the Boctor method according to the results of chapter 5. However, the results do not change in this case when the random clustering is used.

The resulting clusters using Random increase of the vehicle capacity, and R as 1 can be seen in Figure 6.2.

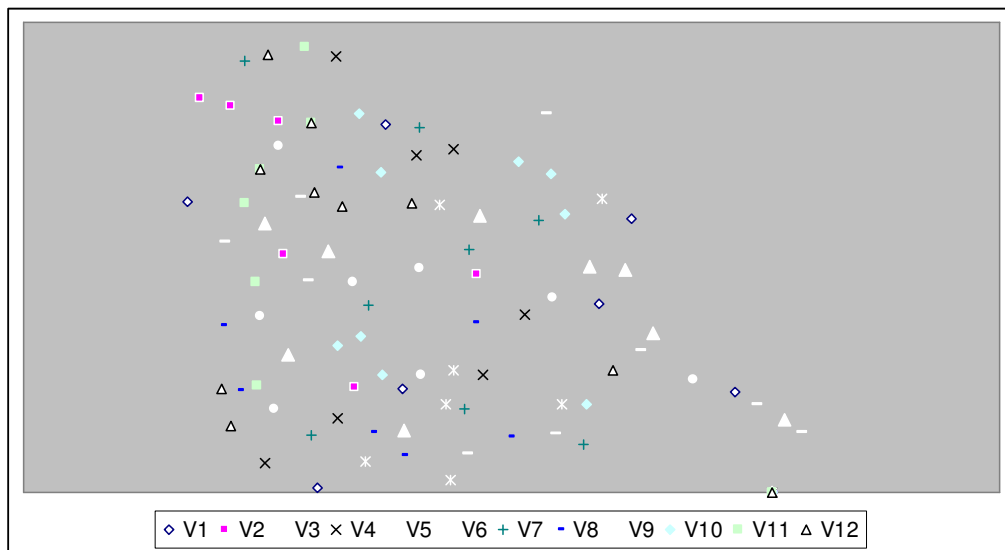


Figure 6. 2 The plot of the resulting clusters, then Random method is used with R=1

The Renaud and Boctor (2002) method is to be improved, and a better user interface is to be designed. The current program we have encoded runs on an MS Dos environment, and all the inputs are in file forms. A user interface that allows the user to specify the distance files can be implemented.

The user interface and further improvements can be implemented after the results are compared to the currently used plan in a real firm. Although Bramel and Simchi-Levi (1997) report that the vehicle routing applications usually result in 15-20% improvement on the costs, the improvement shall be measured.

CHAPTER 7

CONCLUSION

A model based on the HVRP formulation is analyzed for solving the personnel pickup and delivery operations of a firm, in this thesis. The personnel are to be collected with buses having different capacities. The aim is to generate good routes that do not to exceed the maximum travel time allowed very much, while minimizing the total cost of vehicles. The objective function is based on goal programming where the weight of lateness is much less than the cost of a vehicle.

A mathematical model is generated similar to the HVRP model as the basic constraints of our model and the HVRP are the same. The mathematical model developed could not be in stand alone PCs for problem sizes larger than 10, due to memory limitations.

Lower bounds based on the mathematical model developed are calculated loosening the binary variables to integers. Some valid inequalities are used to tighten the lower bounds of HVRP model are also used.

Different heuristics are investigated in order to generate solutions on two different problem sets. The first problem set consists of customers that are delivered randomly and the second set consists of clustered customers. The performance of solution methods in two different settings is also analyzed.

The first heuristic analyzed is a route-first cluster-second heuristic, where the optimum route that passes through all the customers is generated at first, then this route is partitioned to smaller routes. The initial route was solved to optimality and a

mathematical model is developed for the partitioning. The mathematical model developed for partitioning could not be solved due to memory limitations.

The second heuristic we have implemented is the cluster-first route-second heuristic. We have implemented the sweep algorithm with slight modifications. A parameter is used to control the radius of the cluster while adding new customers to the current cluster.

The resulting clusters are routed using a genetic algorithm, and the results are compared with the lower bounds generated. A detailed analysis is conducted on the results of the sweep algorithm. The method used to increase the capacities of the vehicles that give the best results is found to be the method proposed by Boctor and Renaud (2002) is demonstrated to give the best results. It has been observed that the HVRP model give better results when the customers are clustered, on the other hand our model gives better results when the customers are delivered randomly.

A genetic algorithm for HVRP is used as the third heuristic. The genetic algorithm uses nearest neighbor heuristic as the crossover operator, and node insertion as the mutation operator. The results of the genetic algorithm are better compared to the results of the sweep method, and we have demonstrated that the genetic algorithm is capable of improving the results of the sweep algorithm 14%.

The last heuristic we have implemented is a distributed artificial intelligence method. To the best of our knowledge this study includes the first attempt to solve HVRP instances using a multi-agent system. The results are not comparable to problem specific heuristics or the ones presented in this study. However, the heuristics used as decision mechanisms in the agents are very simple.

The demand and vehicle capacities proposed by Golden et al (1984) are used to test the performance of the heuristics. The results obtained by the heuristics described are poor compared to the results of the studies in the literature. The vehicle utilization on the best-known solution of the instances of Golden et al (1984) are demonstrated to be very high, thus alternative good solutions are not found very easily. All the heuristics explained need to be improved, by some improvement mechanisms so that the results they provide become useful.

A simple program that can be used to handle the routing tasks, based on the sweep and genetic algorithms, is generated for the firm to be able to compare the performance of our algorithms with the clustering and routing that is currently used by the firm. A randomly generated problem with the size of the real one is investigated to demonstrate the results on a similar problem. The implementation and comparison of the results is not considered in this study.

For further research, an improvement mechanism that allows inter-cluster movements must be employed, to improve the current results of the heuristics described, in order to come up with solutions good enough to complete the ones reported in the literature.

The mathematical formulation of our model can also be investigated in detail to come generate tight lower bounds for the problem, and to ease the effort of solution of the problem to optimality if possible.

The sweep algorithm can be improved to include tabu search, to incorporate intelligence in generating the cluster structure. Some other clustering methods like k-means can be considered to compare the performance of different clustering techniques. The sweep algorithms can be extended to consider the rotation of the ray in clock-wise to analyze the behavior of the clustering in reverse order.

The power of the genetic algorithm to explore the search space can be increased by using different improvement heuristics such that some promising individual are generated. A more complicated two stage genetic algorithm that clusters the customers first and routes these next can be used to make benefit of the exploitative power of the genetic algorithm.

More intelligent agents need to be considered to improve the results of the multi-agent approach to the HVRP. The central agent can be improved to solve mathematical models to find the best assignment among different vehicle agents in each bid. Moreover the vehicle agents can be improved to use more complicated bidding mechanisms instead of exaggerating the savings.

More runs on random problems are required in order to generalize the behaviour of all algorithms.

The extension of the model to allow the splitting of the demand in busses can also be considered to analyze different properties of the model in detail.

REFERENCES

1. **APPLEGATE, D.** (2007). *World Traveling Salesman Problem*. Georgia Institute of Technology. Retrieved August 3, 2007 from <http://www.tsp.gatech.edu/world>.
2. **APPLEGATE, D., BIXBY, R., CHVÁTAL, V., COOK, W.** (1998). On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM. III: 645 – 656.
3. **ALTINKEMER, K., GAVISH, B.** (1991). Parallel savings based heuristic for the delivery problem. *Operations Research*. 39: 465 – 469.
4. **BEASLEY, J.E., LUCENA, A., POGGI DE ARAGÃO, M.** (2002). The vehicle routing problem. *Handbook of Applied Optimization* Eds. P. M. Pardalos, M. G. C. Resende. Oxford University Press, NY. 17(2): 584 – 594.
5. **BOUDALI, I. FKI, W. AND GHEDIRA, K.** (2004). How to deal with the VRPTW by using multi-agent coalitions”. *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems (HIS’04)*, IEEE Computer Society.
6. **BOUTILIER C., AND BRAFMAN R. I.** (1997). Planning with concurrent interacting actions. *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*. (Providence, Rhode Island). AAAI Press/MIT Press. 720 – 726.
7. **BOWLING, M., JENSEN, R., VELOSO, M. M.** (2002). A formalization of equilibria for multiagent planning, *Proceedings of the AAAI-2002 Workshop on Multiagent Planning*.
8. **BRAMEL, J., SIMCHI-LEVI, D.** (1995). A location based heuristic for general routing problems. *Operations Research*. 43: 649 – 660.
9. **BRAMEL, J., SIMCHI-LEVI, D.** (1997). A case study: school bus routing. *Logic of Logistics: Theory, Algorithms & Applications for Logistics Management*. Springer series in Operations Research. NY. 14: 239 – 254.

10. **BREMERMANN, H. J., ROGSON, M., AND SALAFF, S.** (1966). Global properties of evolution processes. *Natural Automata and Useful Simulations*. Eds H. H. Pattec, E. A. Edelsack, L. Fein, and A. B. Callahan, Washington, DC: Spartan, 1: 3 – 41.
11. **BULLNHEIMER, B., HARTL, R.F., STRAUSS, C.** (1999). Applying the ant system to the vehicle routing problem. Ed. Voss, S., Martello, S., Osman, I.H., Roucairol, C., *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer. Boston. MA. 109 – 120.
12. **ČERNÝ, V.** (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory Applications* (45): 41-51.
13. **ÇETINKAYA, F. C.** (2007). *İngilizce – Türkçe Endüstri Mühendisliği ve Mühendislik Yönetimi Terimleri Sözlüğü* (A Dictionary of Industrial Engineering and Engineering Management Terms). MMO, Ankara.
14. **CHRISTOFIDES, N., MINGOZZI, A., TOTH, R.** (1979). The vehicle routing problems. Ed. Christofides, N., Mingozzi, A., Toth, R., Sandi, C., *Combinatorial Optimization*. Wiley, Chichester. 315 – 338.
15. **CHOI, E., TCHA, D.W.,** (2005). A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*. 34: 2080 – 2095.
16. **CLARKE, G., WRIGHT, J. V.** (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*. 12: 568 – 581.
17. **COLORNI, A., DORIGO, M., MANIEZZO, V.** (1991) Distributed optimization by ant colonies. Ed. Varela, F., Bourguine, P., *Proceedings of European Conference on Artificial Life*, Elsevier, Amsterdam.
18. **DANTZIG, G.B., RAMSER, K.H.** (1959). The truck dispatching problem. *Operations Research*. 12: 80 – 91.
19. **DEMİR, E.** (2004). *Analysis of Evolutionary Algorithms for Constrained Routings Problems.*, MSc Thesis, Ankara, METU.
20. **DESROCHERS, M., LAPORTE, G.** (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*. 10: 27-36.

21. **DESROCHERS, M., LENSTRA, J. K., SAVELSBERGH, M. W. P.** (1990). A classification for vehicle routing problem and scheduling problems. *Journal of Operational Research Society*. 46: 322 – 332.
22. **DESROCHERS, M., VERHOOG, T.W.** (1989). *A matching based savings heuristic algorithms for the vehicle routing problem*. Technical Report. Cahiers du GERAD G-89-04. Ecole des Hautes Etudes Commerciales de Montréal.
23. **DESROCHERS, M., VERHOOG, T.W.** (1991). A new heuristic for the fleet size and mix vehicle routing problem. *Computers and Operational Research*. 18-3:263 – 274.
24. **DONDO, R., MÉNDEZ, C.A., CERDÁ, J.** (2003). An optimal approach to the multi depot heterogeneous vehicle routing problem with time window and capacity constraints. *Latin American Applied Research*. 33: 129 – 134.
25. **ERDOĞDU, U.** (2004) *Resource Based Plan Revision In Dynamic Multi-Agent Environments*. (Unpublished) Master's Thesis, Middle East Technical University, Department of Computer Engineering.
26. **EPHRATI E., ROSENSCHEIN, J. S.** (1997). A heuristic technique for multiagent planning. *Annals of Mathematics and Artificial Intelligence*. 20(1-4): 13 – 67.
27. **FISCHETTI, M., TOTH, P., VIGO, D.** (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*. 42: 846 – 859.
28. **FISCHER, K., CHAIB-DRAA, B., MÜLLER, J. P., PISCHEL, M., GERBER, C.** (1999) A simulation approach based on negotiation and cooperation between agents: a case study, *IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews*, Vol. 29, No. 4.
29. **FISHER, M. L., JAIKUMAR, R.** (1981). A generalized assignment heuristic for the vehicle routing problem. *Networks*. 11: 109 – 124.
30. **GENCER, C., TOP, İ., AYDOGAN, E.K.** (2006). A new intuitional algorithm for solving heterogeneous fixed fleet vehicle routing problems. Passenger pickup algorithm. *Applied Mathematics and Computation*. 181: 1552-1567.
31. **GENDREAU, M., HERTZ, A., LAPORTE, G.** (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*. 40: 1276 – 1290
32. **GENDREAU, M., LAPORTE, G., MUSARAGANYI, C., TAILLARD, E.D.** (1999). A tabu search heuristic for the heterogeneous vehicle routing problem. *Computers and Operations Research*. 26: 1153 – 1173.

33. **GENDREAU, M., LAPORTE, G., POTVIN, J. Y.** (2001). Metaheuristics for the capacitated VRP. *The Vehicle Routing Problem*, Ed. Toth, P., Vigo, D., SIAM monographs on discrete mathematics. Society of Industrial and Applied Mathematics. Philadelphia. 129 – 154.
34. **GHEYSENS, F., GOLDEN, B., ASSAD, A.** (1984). A comparison of techniques for solving the fleet size and mix vehicle routing problems. *OR Spectrum*. 6: 207 – 216.
35. **GHEYSENS, F., GOLDEN, B., ASSAD, A.** (1986). A new heuristic for determining fleet size composition. *Mathematical Programming Study*. 26: 233 – 236.
36. **GILLET, B.E., MILLER, L.R.** (1974). A heuristic algorithm for the vehicle dispatch problem. *Operations Research*. 22: 340 – 349.
37. **GLOVER, F.** (1986) Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533-549.
38. **GOLDBERG, D.E.** (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company. Reading, MA.
39. **GOLDEN, B.L., ASSAD, A.A.,** (1988). *Vehicle Routing: Methods and Studies*. North Holland, Amsterdam.
40. **GOLDEN, B.L., ASSAD, A.A., LEVY, L., GHEYSENS, F.** (1984). The fleet mix vehicle routing problem. *Computers and Operations Research*. 11: 49 – 66.
41. **GOLDEN, B.L., WASIL, E.A., KELLY, J.P., CHAO, I.M.** (1998). Metaheuristics in vehicle routing. *Management and Logistics* Eds. Crainic, T. G., Laporte, G. Fleet, Kluwer, Boston, MA. 33 – 56.
42. **GOULD, J.** (1969). The size and composition of a road transport fleet. *Operations Research Quarterly*. 20: 81 – 92.
43. **HOLLAND, J.** (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Harbor, MI.
44. **JONKER, R, VOLGENANT, T.** (1984). Non-optimal edges for symmetric traveling salesman problem. *Operations Research*. 32: 837 – 846.
45. **KIRKPATRICK, S., GELATT JR., C.D., VECCHI, M.P.** (1983) Optimization by simulated annealing. *Science*. 220 (4598): 671 - 680
46. **LAPORTE, G., NOBERT, Y.** (1987). Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31: 147 – 184.
47. **LAPORTE, G., OSMAN, I.H.** (1995). Routing problems: a bibliography. *Annals of Operations Research*, 61: 227 – 262.

48. **LAPORTE, G., SEMET, F.** (2001). Classical Heuristics for the Capacitated VRP. *The Vehicle Routing Problem*, Ed. Toth, P., Vigo, D., SIAM monographs on discrete mathematics. Society of Industrial and Applied Mathematics. Philadelphia. 29 – 51.
49. **LARRAÑAGA, P., KUIJPERS, C.M.H., MURGA, R.H., INZA, I., & DIZDAREVIC, S.** (1999). Genetic algorithms for the travelling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, 13, 129–170.
50. **LENSTRA, J.K., RINNOOY KAN, A.H.** (1981). Complexity of vehicle routing and scheduling problems. Summaries of Discussion Sessions. *Networks*.11(2): 221 – 227
51. **LI, F., GOLDEN, B., WASIL, E.** (2007). A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 34: 2734 – 2742.
52. **LIMA, C.M.R.R., GOLDBARG, M.C., GOLDBARG, E.F.G.** (2004). A memetic algorithm for heterogeneous fleet vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 18: 171 – 176.
53. **LIN, S., KERNIGHAN, B.W.** (1973). An effective heuristic algorithm for the traveling salesman problem. *Mathematical Programming*, 10: 367 – 378.
54. **LIU, F.H., SHEN, S.Y.** (1999).The Fleet Size and Mix Vehicle Routing Problem with Time Windows. *The Journal of the Operational Research Society*, 50(7): 721-732.
55. **MCCULLOCH, W. S., PITTS, W.** (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5: 115 – 133.
56. **METROPOLIS, N., ROSENBLUTH, A.W., ROSENBLUTH, M.N. , TELLER, A.H. AND TELLER E.** (1953). Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087-1092.
57. **MICHAILEWITZ, Z., FOGEL, D.B.** (2003). *How to Solve It: Modern Heuristics*. Germany Springer-Verlag
58. **MILLER, C., TUCKER, A., ZEMLIN, R.** (1960). Integer programming formulations and traveling salesman problems. *Journal of ACM*. 7: 326 – 329.
59. **NAGATA, Y., KOBAYASHI, S.** (1997) Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem. *Proceedings of the 7th International Conference on Genetic Algorithms*, 450-457.
60. **OCHI, L.S., VIANNA, D.S., DRUMMOND, L.M.A., VICTOR, A.O.** (1998a) An Evolutionary Hybrid Metaheuristic for Solving the Vehicle

Routing Problem with Heterogeneous Fleet. *Genetic Programming*. Lecture Notes in Computer Science. 1391: 187 -194.

61. **OCHI, L.S., VIANNA, D.S., DRUMMOND, L.M.A., VICTOR, A.O.** (1998b). A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Future Generation Computer Systems*. 14: 282 – 292.
62. **ÖNDER, İ.** (2007). *A Genetic Algorithm for TSP with Backhauls Based on Conventional Heuristics*. (Unpublished). Master's Thesis, Middle East Technical University, Department of Information Systems.
63. **OR, I.** (1976). *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*. Ph.D. dissertation, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.
64. **OSMAN, I. H.** (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41: 421 – 425.
65. **OSMAN, I. H., LAPORTE, G.** (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, Volume 63. 511-623.
66. **OSMAN I. H., SALHI S.** (1994). *Local search strategies for the Vehicle Fleet Mix Problem*, Institute of Mathematics and Statistics, University of Kent, Canterbury.
67. **RECHENBERG, I.** (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart, Germany: Frommann-Holzboog.
68. **REINELT, G.** (1996). *The Travelling Salesman Problem: Computational Solutions for TSP Applications*. Lecture Notes in Computer Science Ed. Goos, G., Hartmanis, J.. Springer Verlag.
69. **REINELT, G.** (2007). *TSPLIB*. Retrieved 10 August 2007 from <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>
70. **RENAUD, J., BOCTOR, F.F.** (2002). A sweep-based algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research*, 140: 618 – 628.
71. **ROCHAT, Y., TAILLARD, E.D.** (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1: 147 – 167.
72. **SALHI, S., RAND, G.K.** (1993). Incorporating vehicle routing into the vehicle fleet decomposition problem. *European Journal of Operations Research*, 66: 313-330.

73. **SALHI, S., SARI, M., SAIDI, D., TOUATI, N.** (1992). Adaptation of some vehicle mix heuristics. *Omega*. 20: 653 – 660.
74. **SEYRAN, İ.**, (2006). *An application of the Vehicle Routing Problem to a Glass Manufacturing Firm.* (Unpublished). Master's Thesis, Çankaya University, Department of Industrial Engineering.
75. **SOLOMON, M. M.**, (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35 (2): 254 – 265.
76. **TAILLARD, E. D.** (1993). Parallel iterative search methods for vehicle routing problem. *Networks*, 23: 661 – 673.
77. **TAILLARD, E.D.** (1996). *A heuristic column generation method for the heterogeneous fleet VRP.* Publication CRT – 96 – 03, Centre de Recherche sur les transports, Université de Montréal.
78. **TAILLARD, E.D.** (1999). A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO – Operations Research*. 33: 1-14.
79. **THANGIAH, S.R., SHMYGELSKA, O., MENNELL, W.** (2001). An agent architecture for vehicle routing problems. *Proceedings of the Association for Computing Machinery's Symposium on Applied Computing*, Las Vegas, U.S.A., 517-521.
80. **TOTH, P., VIGO, D.** (2001a). An Overview of the vehicle routing problems. *The Vehicle Routing Problem*, Ed. Toth, P., Vigo, D., SIAM monographs on discrete mathematics. Society of Industrial and Applied Mathematics. Philadelphia.1 – 27.
81. **TOTH, P., VIGO, D.** (2001b). Branch-and-bound algorithms for the capacitated VRP. *The Vehicle Routing Problem*, Ed. Toth, P., Vigo, D., SIAM monographs on discrete mathematics. Society of Industrial and Applied Mathematics. Philadelphia. 29 – 51.
82. **WARK, P., HOLT, J.** (1994). A repeated matching heuristic for the vehicle routing problem. *Journal of Operations Research Society*. 45: 1156, 1167.
83. **WASSAN, N.A., OSMAN, I.H.** (2002). Tabu Search variants for the mix fleet vehicle routing problem. *Journal of Operations Research Society*. 53: 768 – 782.
84. **WHITLEY, D. STARKWEATHER, T., D'ANN F.** (1989). Scheduling Problems and Travelling Salesman: The Genetic Edge Recombination Operator. *Proceedings on the Third International Conference on Genetic Algorithms*. Ed. Schaffer, J., 133-140. Los Altos, CA: Morgan Kaufmann Publishers.
85. **WREN, A.** (1971). *Computers in Transport Planning and Operation*. Ian Allan, London.

86. **YAMAN, H.** (2006). Formulations and Valid Inequalities for the Heterogeneous Vehicle Routing Problem. *Mathematical Programming*. A-106: 365 – 390.

APPENDIX A

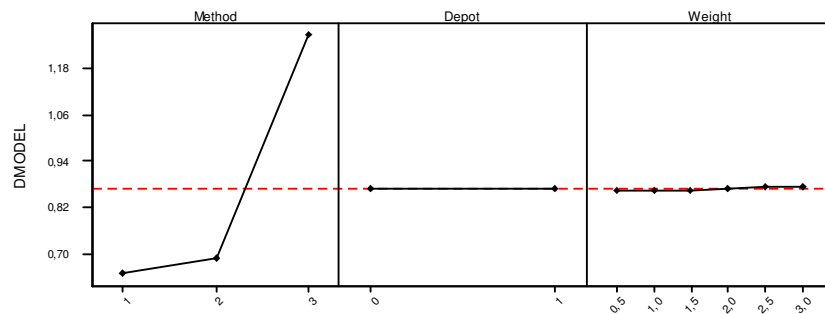
General Linear Model: DMODEL versus Size; Method; Depot; Weight

Factor	Type	Levels	Values
Size	fixed	4	20 50 75 100
Method	fixed	3	1 2 3
Depot	fixed	2	0 1
Weight	fixed	6	0,5 1,0 1,5 2,0 2,5 3,0

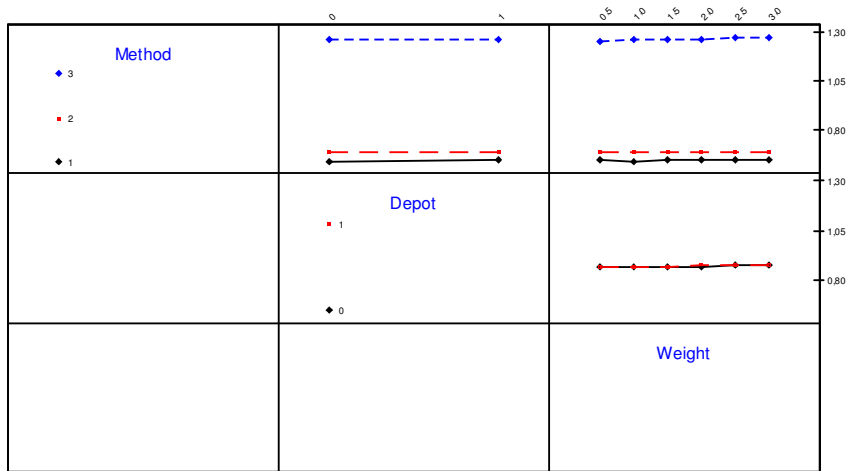
Analysis of Variance for DMODEL, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Size	3	116,2128	116,2128	38,7376	138,17	0,000
Method	2	68,2784	68,2784	34,1392	121,77	0,000
Depot	1	0,0012	0,0012	0,0012	0,00	0,948
Weight	5	0,0071	0,0071	0,0014	0,01	1,000
Method*Depot	2	0,0005	0,0005	0,0003	0,00	0,999
Method*Weight	10	0,0128	0,0128	0,0013	0,00	1,000
Depot*Weight	5	0,0008	0,0008	0,0002	0,00	1,000
Method*Depot*Weight	10	0,0027	0,0027	0,0003	0,00	1,000
Error	825	231,2926	231,2926	0,2804		
Total	863	415,8087				

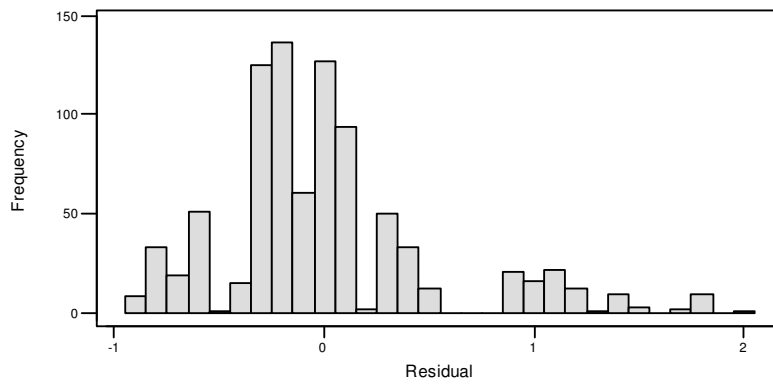
Main Effects Plot - Data Means for DMODEL



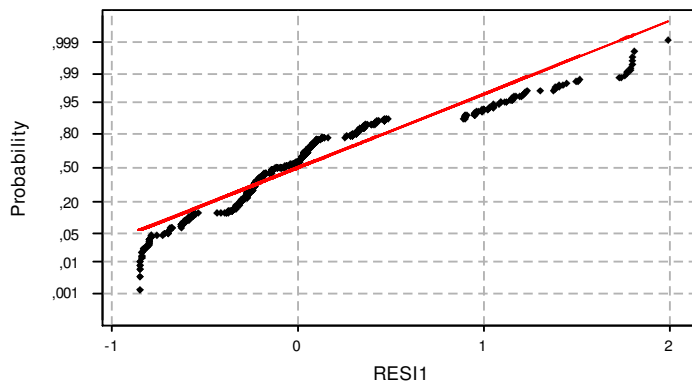
Interaction Plot - Data Means for DMODEL



Histogram of the Residuals
(response is DMODEL)



Normal Probability Plot



Average: -0,0000000
StDev: 0,517697
N: 864

Anderson-Darling Normality Test
A-Squared: 31,652
P-Value: 0,000

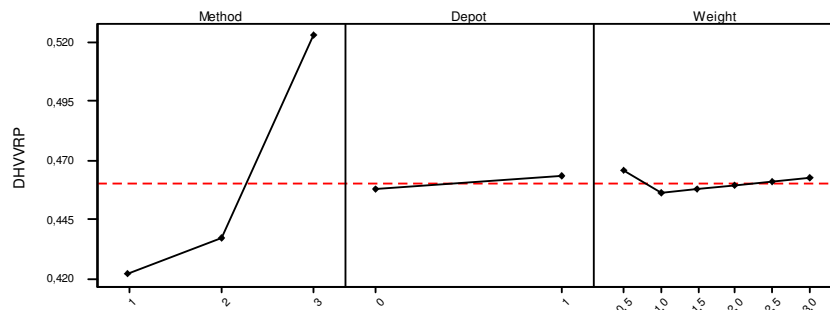
General Linear Model: DHVVRP versus Size; Method; Depot; Weight

Factor	Type	Levels	Values
Size	fixed	4	20 50 75 100
Method	fixed	3	1 2 3
Depot	fixed	2	0 1
Weight	fixed	6	0,5 1,0 1,5 2,0 2,5 3,0

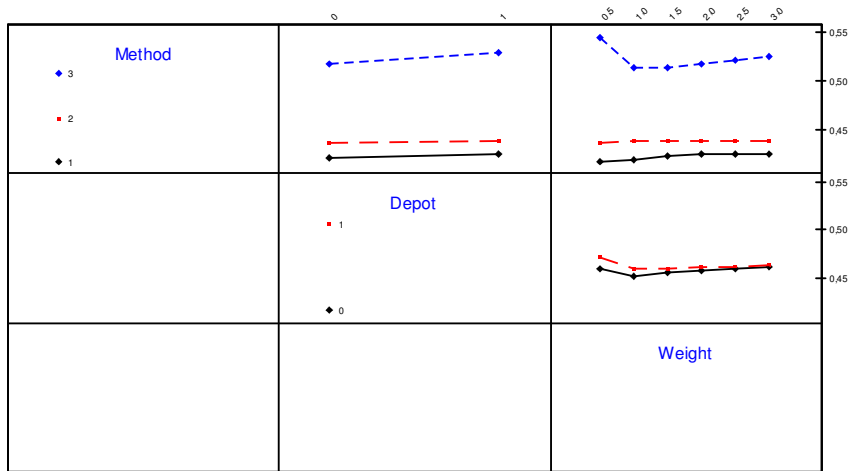
Analysis of Variance for DHVVRP, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Size	3	77,6217	77,6217	25,8739	1020,76	0,000
Method	2	1,7038	1,7038	0,8519	33,61	0,000
Depot	1	0,0069	0,0069	0,0069	0,27	0,602
Weight	5	0,0080	0,0080	0,0016	0,06	0,997
Method*Depot	2	0,0037	0,0037	0,0018	0,07	0,930
Method*Weight	10	0,0296	0,0296	0,0030	0,12	1,000
Depot*Weight	5	0,0023	0,0023	0,0005	0,02	1,000
Error	835	21,1654	21,1654	0,0253		
Total	863	100,5413				

Main Effects Plot - Data Means for DHVVRP

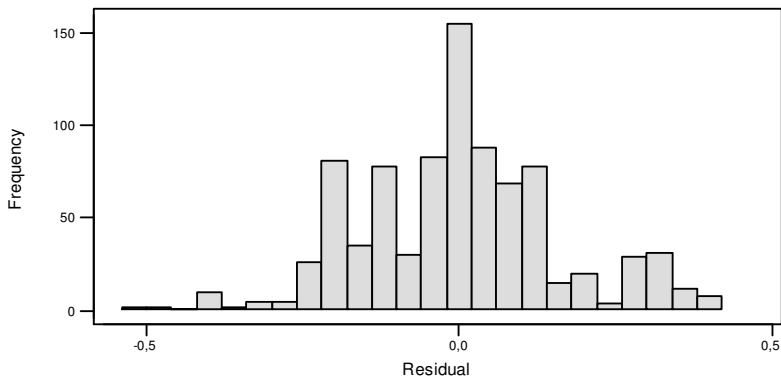


Interaction Plot - Data Means for DHVVRP

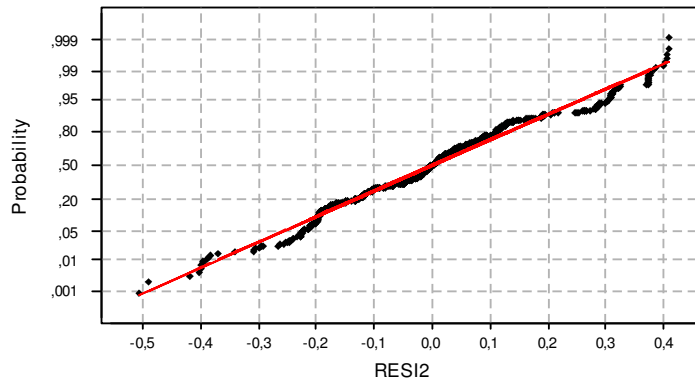


Histogram of the Residuals

(response is DHVVRP)



Normal Probability Plot



Average: 0,0000000
StDev: 0,156606
N: 864

Anderson-Darling Normality Test
A-Squared: 6,168
P-Value: 0,000

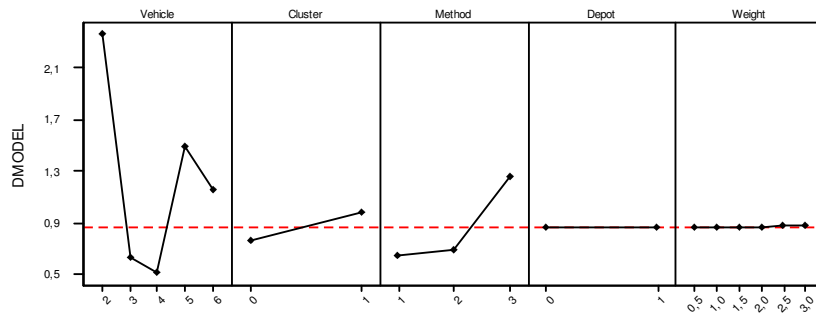
General Linear Model: DMODEL versus Size; Vehicle; ...

Factor	Type	Levels	Values
Size	fixed	4	20 50 75 100
Vehicle	fixed	5	2 3 4 5 6
Cluster	fixed	2	0 1
Method	fixed	3	1 2 3
Depot	fixed	2	0 1
Weight	fixed	6	0,5 1,0 1,5 2,0 2,5 3,0

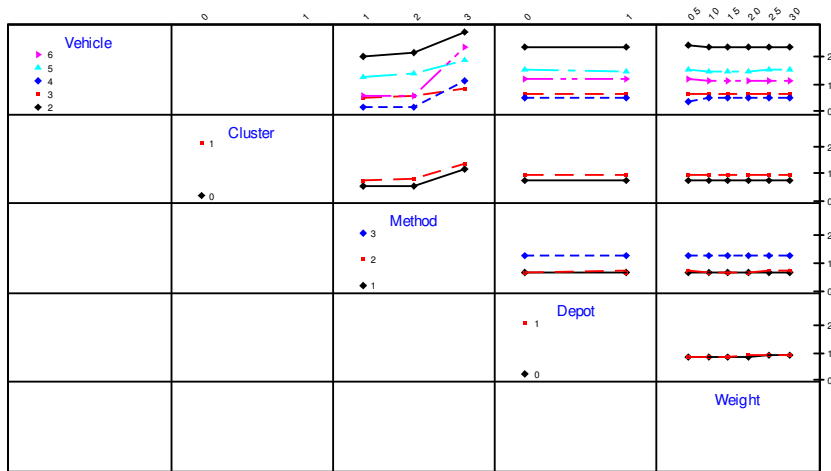
Analysis of Variance for DMODEL, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Size	3	116,213	45,325	15,108	104,57	0,000
Vehicle	4	104,746	98,662	24,665	170,72	0,000
Cluster	1	4,191	4,191	4,191	29,01	0,000
Method	2	68,278	68,278	34,139	236,29	0,000
Depot	1	0,001	0,001	0,001	0,01	0,927
Weight	5	0,007	0,007	0,001	0,01	1,000
Error	847	122,372	122,372	0,144		
Total	863	415,809				

Main Effects Plot - Data Means for DMODEL

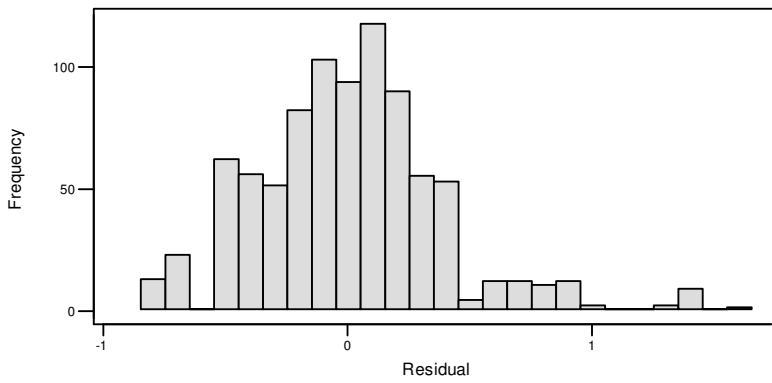


Interaction Plot - Data Means for DMODEL

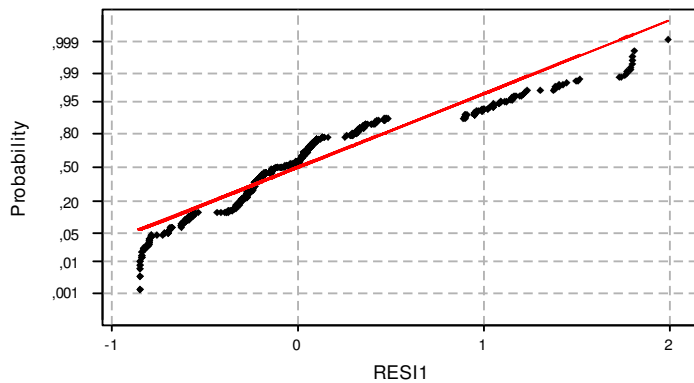


Histogram of the Residuals

(response is DMODEL)



Normal Probability Plot



Average: -0,0000000
StDev: 0,517697
N: 864

Anderson-Darling Normality Test
A-Squared: 31,652
P-Value: 0,000

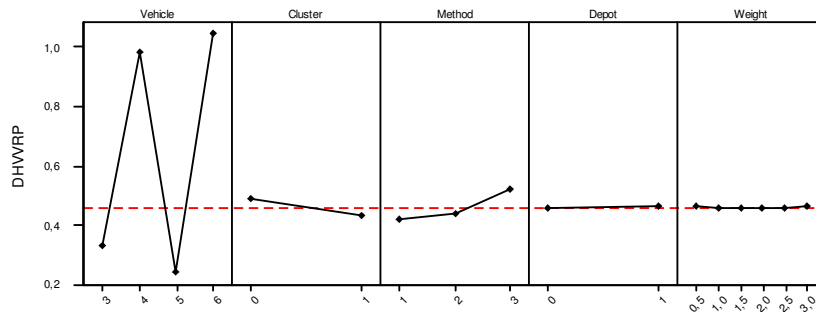
General Linear Model: DHVVRP versus Size; Vehicle; ...

Factor	Type	Levels	Values
Size	fixed	4	20 50 75 100
Vehicle	fixed	4	3 4 5 6
Cluster	fixed	2	0 1
Method	fixed	3	1 2 3
Depot	fixed	2	0 1
Weight	fixed	6	0,5 1,0 1,5 2,0 2,5 3,0

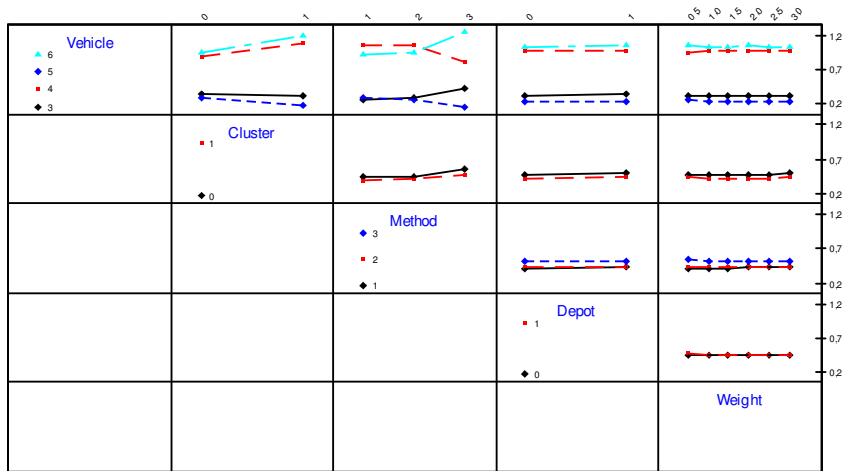
Analysis of Variance for DHVVRP, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Size	3	77,6217	9,5055	3,1685	267,95	0,000
Vehicle	3	5,3766	4,2993	1,4331	121,19	0,000
Cluster	1	0,1679	0,0037	0,0037	0,31	0,576
Method	2	1,7038	0,0539	0,0269	2,28	0,103
Depot	1	0,0069	0,0007	0,0007	0,06	0,808
Weight	5	0,0080	0,0034	0,0007	0,06	0,998
Vehicle*Cluster	3	1,1940	1,1940	0,3980	33,66	0,000
Vehicle*Method	6	4,9342	4,8899	0,8150	68,92	0,000
Vehicle*Depot	3	0,0086	0,0086	0,0029	0,24	0,867
Vehicle*Weight	15	0,0242	0,0249	0,0017	0,14	1,000
Cluster*Method	2	0,0431	0,0431	0,0216	1,82	0,162
Cluster*Depot	1	0,0001	0,0001	0,0001	0,01	0,917
Cluster*Weight	5	0,0041	0,0041	0,0008	0,07	0,997
Method*Depot	2	0,0037	0,0037	0,0018	0,16	0,856
Method*Weight	10	0,0296	0,0296	0,0030	0,25	0,991
Depot*Weight	5	0,0023	0,0023	0,0005	0,04	0,999
Error	796	9,4127	9,4127	0,0118		
Total	863	100,5413				

Main Effects Plot - Data Means for DHVVRP

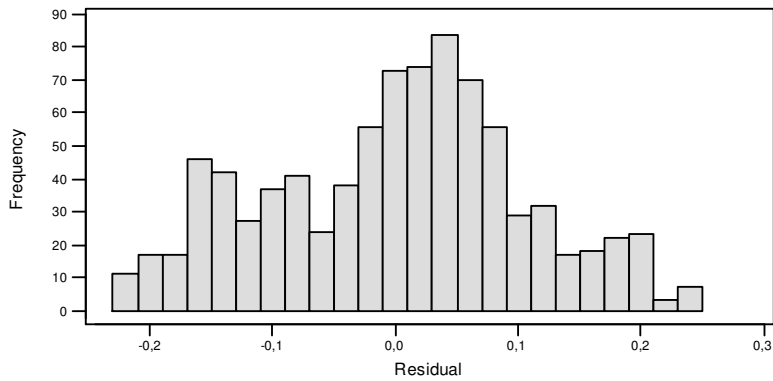


Interaction Plot - Data Means for DHVVRP

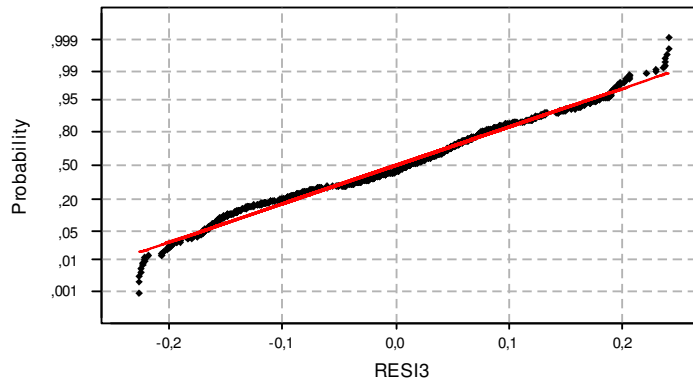


Histogram of the Residuals

(response is DHVVRP)



Normal Probability Plot



Average: 0,0000000
StDev: 0,104436
N: 864

Anderson-Darling Normality Test
A-Squared: 4,482
P-Value: 0,000

APPENDIX B

GAMS Model of LP Relaxation of the Proposed Model

```
SETS
i customers /0*20/
k vehicle /1*3/;
alias(i,j)
File gamsout / d:\soln.txt/;
PARAMETERS
$batinclude d:\data\datas\t4.txt "Q(k)" "c(k)";
$include d:\data\datas\dg4.txt;
TABLE
d(i,j) weight
$include d:\data\datas\gol4.txt ;
VARIABLES
Z;
POSITIVE VARIABLES
u(i)
u(j)
t(i)
t(j)
Lp(i)
x(i,j,k)
a(i,k)
b(i,k);

EQUATIONS
obj
c1(i)
c2(i)
c3(k)
c4(i,k)
c5(i,k)
c6(i,j)
c7(i)
c8(i,j)
c9(i)
c10(i)
c11(i,k)
c12;
obj..Z=e=sum((i,k),c(k)*a(i,k))+ sum((i),1.1*Lp(i));
c1(i)$ (ord(i)>1)..sum((k),a(i,k)+b(i,k))=e=1;
c2(i)$ (ord(i)>1)..u(i)=1-sum((k),Q(k)*(a(i,k)+b(i,k)));
c3(k)..sum((j)$ (ord(j)>1), x("0",j,k))=e=sum((i)$ (ord(i)>1),a(i,k));
```

```

c4(j,k)$ (ord(j)>1)..sum((i), x(i,j,k))=e=a(j,k)+b(j,k);
c5(i,k)$ (ord(i)>1)..sum((j)$ (ord(j)>1), x(i,j,k))=e=b(i,k);
c6(i,j)$ ((ord(i)>1) and (ord(j)>1) and ord(i) ne ord(j))..u(j) =g= u(i) + qa(j) -
    sum((k),Q(k)*(a(i,k)+b(i,k))) + sum((k), Q(k)*x(i,j,k));
c7(i) $ (ord(i)>1)..u(i) =g= qa(i) + sum((j,k),qa(j)* x(j,i,k));
c8(i,j)$ (ord(j)>1).. t(i)+d(i,j)*sum((k),x(i,j,k))-t(j)=l=1000*(1-sum((k),x(i,j,k)));
c9(i)..t(i)+d(i,"0")=l=50*sum((k),a(i,k))+Lp(i)+1000000*(1-sum((k),(a(i,k))));
c10(i).. 10000000*(sum((k),(a(i,k)))) =g= Lp(i);
c11(i,k)..x(i,i,k)=e=0;
c12..sum((i,j,k),qa(i)*x(i,j,k))=l=sum((i,k), Q(k)*a(i,k))
option iterlim =999999999;
OPTION ResLim = 21600;
MODEL tez /ALL/;
SOLVE tez MINIMIZING z USING MIP;
put gamsout;

```

GAMS Model of LP Relaxation of the HVRP Model

```

SETS
i customers /0*20/
k vehicle /1*3/;
alias(i,j);
PARAMETERS
$batinclude d:\data\datas\t4.txt "Q(k)" "c(k)" "w(k)";
$include d:\data\datas\dg4.txt;
TABLE
d(i,j) weight
$include d:\data\datas\gol4.txt;
VARIABLES
Z;
POSITIVE VARIABLES
u(i)
u(j)
x(i,j,k);
BINARY VARIABLES
a(i,k)
b(i,k);
EQUATIONS
obj
c1(i)
c2(i)
c3(k)
c4(i,k)
c5(i,k)
c6(i,j)
c7(i)
c14
c15;
obj..Z=e=sum((i,k),c(k)*a(i,k)+d(i,'0')*a(i,k))+ sum((i,j,k),w(k)*d(i,j)*x(i,j,k));
c1(i)$ (ord(i)>1)..sum((k),a(i,k)+b(i,k))=e=1;
c2(i)$ (ord(i)>1)..u(i)=l=sum((k),Q(k)*(a(i,k)+b(i,k)));
c3(k)..sum((j)$ (ord(j)>1), x("0",j,k))=e=sum((i)$ (ord(i)>1),a(i,k));
c4(j,k)$ (ord(j)>1)..sum((i), x(i,j,k))=e=a(j,k)+b(j,k);
c5(i,k)$ (ord(i)>1)..sum((j)$ (ord(j)>1), x(i,j,k))=e=b(i,k);
c6(i,j)$ ((ord(i)>1) and (ord(j)>1) and ord(i) ne ord(j))..u(j) =g= u(i) + qa(j) -
    sum((k),Q(k)*(a(i,k)+b(i,k))) + sum((k), Q(k)*x(i,j,k));

```

```
c7(i) $(ord(i)>1)..u(i) =g= qa(i) + sum((j,k),qa(j)* x(j,i,k));
c14..ceil(sum((i),qa(i))/10)=l=sum((i,k)$(ord(i)>1),ceil(Q(k)/10)*a(i,k));
c15..sum((i,k),Q(k)*a(i,k))=g=sum((i,k),qa(i)*(a(i,k)+b(i,k)));
MODEL tez /ALL/;
SOLVE tez MINIMIZING z USING MIP;
```

APPENDIX C

```
SETS
i customers /0*20/
t vehicle type /1*5/
k vehicle /1*18/;
alias(i,j)
alias(k,m);
File gamsout /d:\data\res_G_03.txt/;
PARAMETERS
$batinclude d:\data\g03.txt "Q(t)" "c(t)" "qa(i)" "co(i)";
SCALAR
H /5/;
VARIABLES
Z;
POSITIVE VARIABLES
d(i);
BINARY VARIABLES
x(i,k)
y(k,t);
EQUATIONS
obj
c1(i,k)
c2(k)
c3(i)
c5(k)
c6(k);
obj..Z=e=sum((t,k),c(t)*y(k,t))+ sum((i),d(i-1));
c1(i,k)..(co(i)+co(i-1))* (-1+x(i,k)+sum((m)$((ord(m)>ord(k) or(ord(m)<ord(k)))),x(i-1,m)))=l=d(i-1);
c2(k).. sum((t),y(k,t))=l=1;
c3(i)..sum((k),x(i,k))=e=1;
c5(k)..(co('0')+co('20'))*(-1+x('0',k)+sum((m)$((ord(m)>ord(k) or(ord(m)<ord(k)))), x('20',m))) =l=d('0') ;
c6(k).. sum((i), qa(i)*x(i,k))=l=sum((t), Q(t)*y(k,t));
option iterlim =999999999;
OPTION ResLim = 172800;
MODEL tez /ALL/;
SOLVE tez MINIMIZING z USING MIP;
put gamsout;
  scalar col column number /18/ ;
  loop(k, put @col k.tl; col=col+10;);
  put /; col=10;
  loop(i,
    loop (k, put @col x.l(i,k):10:1, @0 i.tl ; col=col+10; ) ;
    put / ; col=10 ;
  );
put /;
loop(t,
```



```
    loop (k, put @col y.l(k,t):10:1, @0 t.tl ; col=col+10; ) ;  
    put / ; col=10 ;  
);
```

APPENDIX D

The C codes of all CFRS algorithms are provided in the Appendix CD, attached on the back cover of the thesis. Sample CFRS algorithm code, where the ochi method is used is given below.

CFRS C code for Ochi Method

```
#include<iostream.h>
#include<iomanip.h>
#include<stdlib.h>
#include<time.h>
#include<fstream>
#include<math.h>
#include<string.h>
#include<time.h>
void readdistances(char * filename);
int sweep(int * liste, double t,int N, char * fname);
int * sort(int * liste, int N);
int demand[3500],capacity;
double dist[3500][3500],x[3500],y[3500];
int capacity[10],cno,yazdir[3500];
double cost[10];
char boy [7];
int KalanArac[10];
double RadiusRatio;
int uzunluk,bir;
char * fname;
int main(int argc, char **argv)
{
    int i,j,k,semp;
    char dosyadi[20], * num,NofTrucks;
    fname= new char [20];
    int * sortedcities;
    for(i=0;i<20;i++)
        dosyadi[i] = argv[1][i];
    RadiusRatio=atof(argv[2]);
    bir=atoi(argv[3]);
    semp=strlen(dosyadi);
    fname=strcpy(fname,dosyadi);
    fname[semp-4]=fname[semp];
    FILE *finput;
    FILE * foutput;

    finput=fopen(dosyadi,"r");
    for(i=0;i<10;i++)
```

```

{
    capacity[i]=-1;
}
readdistances(dosyadi);
boy[0]='A';
boy[1]='B';
boy[2]='C';
boy[3]='D';
boy[4]='E';
boy[5]='F';
boy[6]='Y';
uzunluk--;
sortedcities= new int[uzunluk+1];
sortedcities=sort(sortedcities,uzunluk);
if (bir>uzunluk)
    bir=0;

NofTrucks=sweep(sortedcities,RadiusRatio,uzunluk,dosyadi);
delete sortedcities;
return 0;
}

```

```

void readdistances(char * filename)
{
    double result,hold;
    int i,j,k;
    FILE *fread;
    fread = fopen(filename,"r");
    printf("dosyayy actim, %s\n",filename);
    fscanf(fread,"%d %d",&uzunluk, &cno);
    for(i=0;i<cno;i++)
    {
        fscanf(fread,"%d",&capacity[i]);
        fscanf(fread,"%lf",&cost[i]);
        KalanArac[i]=100;
    }
    for(i=0;i<uzunluk;i++)
    {
        fscanf(fread,"%d %lf %lf\n",&k,&x[i],&y[i]);
    }
    for(i=0;i<uzunluk;i++)
    {
        fscanf(fread," %d %d\n",&k,&demand[i]);
    }
    fclose(fread);
    for(i=0;i<uzunluk;i++)
        for(j=0;j<uzunluk;j++)
        {
            if(i==j)
                dist[i][j]=0;
            else
            {
                hold=sqrt(pow((x[i]-x[j]),2) + pow((y[i]-y[j]),2));
                if(hold-floor(hold)<0.5)
                    dist[i][j]=floor(hold);
                else dist[i][j]=floor(hold+1);
            }
        }
    }
}

```

```

    }
}

int * sort(int * liste, int N)
{
    int i,j,k,tut,temp,flag=0;
    for(i=0;i<=N;i++)
        liste[i]=0;
    for(i=1;i<N+1;i++)
    {
        if((x[i]-x[0])>0)
        {
            tut=i;
            for(j=0;j<i-1;j++)
            {
                if((x[tut]-x[0])>(x[liste[j]]-x[0]))
                {
                    temp=liste[j];
                    liste[j]=tut;
                    tut=temp;
                }
                if(tut==0)
                {
                    flag=1;
                    break;
                }
            }
            if(flag==0)
                liste[j]=tut;
        }
        else
        {
            tut=i;
            for(j=N-1;j>N-i;j--)
            {
                if((x[tut]-x[0])<=(x[liste[j]]-x[0]))
                {
                    temp=liste[j];
                    liste[j]=tut;
                    tut=temp;
                }
                if(tut==0)
                {
                    flag=1;
                    break;
                }
            }
            if(flag==0)
                liste[j]=tut;
        }
        flag=0;
    }
    for(i=0;i<N;i++)
        printf("%d ", liste[i]);
}

```

```

        return liste;
    }

int sweep(int * liste, double t,int N, char * fname)
{
    int i,j,k, count=0,tount=0,doluluk=0,dlag=0,positives,n=0,semp=0,capp=0,c,  buyuk,ilk,cik,
    tapcapacity, tapp,z;
    double temp,dist,radius, centerx,centery,eskix,eskiy, totalcost=0,sayy;
    bool * check;
    int * current;
    char * name ;
    name=new char [20];
    name[0]=NULL;
    char * tame ;
    char tut [20];
    clock_t start,end;
    name[0] ='t';
    name[1] ='e';
    name[2] ='m';
    name[3] ='p';
    name[4]=NULL;
    FILE * fent;
    FILE *fcent;

    check=new bool [N+1];
    current= new int [N+1];
    for(i=0;i<N+1;i++)
    {
        check[i]=false;
        current[i]=-1;
    }
    fcent=fopen("route_res.txt","a");
    fprintf(fcent,"%lf: %d: %d:",RadiusRatio,cno, bir);
        if(y[bir]>0)
        {
            positives =1;
        }
        else positives =0;
    start=clock();
    do
    {
        capp=0;
        capcapacity=capacity[capp];
        cik=0;
        if(y[bir]>0)
        {
            positives =1;
        }
        else positives =0;
    for(i=bir;((i<N) || (cik<2));i++)
    {
        if((i==N)&&(positives==1))
        {
            i=0;
            positives=0;
            cik++;
        }
    }
}

```

```

else if(i==N)
{
    i=0;
    positives=1;
    cik++;
}
if((i==bir) && ((positives==1)&&(y[bir]>0)||positives==0)&&(y[bir]<0))
{
    centerx=x[0];
    centery=y[0];
}
tapcapacity=capcapacity;
tapp=capp;
if((demand[liste[i]]+doluluk)>capcapacity)
{
    buyuk=-1;
    ilk=-1;
    for(c=0;c<cno;c++)
    {
        if((demand[liste[i]]+doluluk<capacity[c]) && (buyuk<0))
            buyuk=c;
        if((demand[liste[i]]<capacity[c])&&(ilk<0))
            ilk=c;
        if((ilk>=0)&&(buyuk>=0))
            break;
    }
    if((ilk>=0)&&(cost[capp]*(capacity[capp]-doluluk)+cost[ilk]*(capacity[ilk]-demand[liste[i]])>(cost[buyuk]*(capacity[buyuk]-demand[liste[i]]+doluluk)) && (buyuk>=0))
    {
        tapcapacity=capacity[buyuk];
        tapp=buyuk;
    }
}

if((((y[liste[i]]-y[0]>=0) && (positives==1)) || ((y[liste[i]]-y[0]<0) && (positives==0))) && (check[liste[i]]==false) && (demand[liste[i]]+doluluk<tapcapacity))
{
    capcapacity=tapcapacity;
    capp=tapp;

    temp=sqrt((centerx-x[liste[i]])*(centerx-x[liste[i]])+(centery-y[liste[i]])*(centery-y[liste[i]]));

    if((dlag==0) || (radius > temp))
    {
        current[tount]=liste[i];
        tount++;
        count++;
        if(doluluk!=0)
        {
            if(tount==1)
            {
                centerx=eskix;
                centery=eskiy;
            }
        }
    }

    centerx=(centerx*doluluk+x[liste[i]]*demand[liste[i]])/(doluluk+demand[liste[i]]);

```

```

centery=(centery*doluluk+y[liste[i]]*demand[liste[i]])/(doluluk+demand[liste[i]]);
//      centerx=(centerx+x[0])/2;
//      centery=(centery+y[0])/2;
}
else
{
    centerx= (centerx + x[liste[i]])/2;
    eskix = x[liste[i]];
    centery=(centery + y[liste[i]])/2;
    eskiy = y[liste[i]];
}
if(tount>0)
    dlag=1;
    dist=0;
    for(j=0;j<tount;j++)
    {
        temp=sqrt(pow((centerx-x[current[j]]),2)+pow((centery-
y[current[j]]),2));
        if(dist < temp)
            dist=temp;
    }
    doluluk=doluluk+demand[liste[i]];
    radius = dist * capacity[cno-1] / doluluk * t;

    check[liste[i]]=true;
}
}

if(tount>0)
n++;
tame=strncat(name,itoa(n,tut,10),100);
tame=strncat(tame,".txt",4);
printf("%s",tame);
if(tount>0)
fent=fopen(tame,"w");
semp=strlen(tame);
tame[semp-5-int(log10(n))]=tame[semp];

while(capacity[capp-1]>doluluk)
{
    capp--;
}
if(capp<0)
    capp=0;
KalanArac[capp]--;
if(KalanArac[capp]<=0)
{
    for(z=0;z+capp<=10;z++)
    {
        capacity[capp+z]=capacity[capp+z+1];
        cost[capp+z]=cost[capp+z+1];
        if(capacity[capp+z]==-1)
            break;
    }
    cno--;
}

```

```

totalcost=cost[capp]+totalcost;
if(tount>0)
{
printf("\n%d. kamyon",n);
fprintf(fent,"%d %d %c %lf %d\n", tount+1, bir, boy[capp], cost[capp], doluluk);
fprintf(fent,"0 %lf %lf\n",x[0],y[0]);
fprintf(fcent,"0 ");
for(i=0;i<N;i++)
{
if(current[i]==-1)
break;
printf(" %d (%d)", current[i], demand[current[i]]);
fprintf(fent,"%d %lf %lf\n", current[i],x[current[i]],y[current[i]]);
fprintf(fcent,"%d ",current[i]);
current[i]=-1;
}
fclose(fent);
printf("cost: %lf", cost[capp]);
}
dlag=0;
capp=0;
tount=0;
doluluk=0;
}
while(count<N);
fent=fopen("temp.txt","w");
fprintf(fcent,":%d: %lf: %lf",n, totalcost,double(clock()-start)/CLOCKS_PER_SEC);
fprintf(fcent,"\n");
fprintf(fent,"%d %lf %lf %s %lf %d",n,
start)/CLOCKS_PER_SEC,totalcost, fname, RadiusRatio, bir );
fprintf(fent,"\n");
fclose(fent);
fclose (fcent);
fent=fopen("run_BWD.bat","a");
for(i=0;i<n;i++)
{
fprintf(fent,"Rotate temp%d.txt\n",i+1);
}
fprintf(fent,"Summarize \nStartT");
delete check;
delete current;
return n;
}

```


APPENDIX E

The C codes of the genetic algorithm are also provided in the Appendix CD, attached on the back cover of the thesis.

C code of the Generic Algorithm

```
#include<iostream.h>
#include<iomanip.h>
#include<stdlib.h>
#include<time.h>
#include<fstream>
#include<math.h>
#include<string.h>
#include<time.h>
using std::ifstream;
#define MODLUS 2147483647
#define MULT1 24112
#define MULT2 26143
struct node{
    int plc;
    struct node *nextnode;
};
struct nodek{
    int plc;
    struct nodek * nextnode;
    int * parents;
};
struct list {
    bool chk;
    struct list *nextl;
};
struct nodelist{
    struct node tour;
    struct nodelist * nexttour;
};
struct llist{
    int no;
    double fitness;
    struct node * body;
    struct llist * prev;
    struct llist * next;
};
typedef struct node nodetype;
typedef nodetype *nodeptr;

struct node bir;
```

```

int k=3,l,cr,shortest;
void initializecheck ( struct list *k);
void generate(struct node *bir, int N, bool * bak);
void readdistances(char * filename);
double * assignfitness (double * fitns,struct node * input,int lenght);
void InitializeStat (struct node * one, struct node * input,int count,double *fitns,int *parents,int N);
void selectrand (struct node * one, struct node * input,int count,double *fitns,int *parents,int N);
void yazdir(struct node t,int k);
struct node * createunion(struct node *,int count,int N, struct node * liste);
struct node * writetolist (struct node *liste,int count,int previous,int first);
void deletelist (struct nodelist * liste);
void listesil (struct node * girilen, int N);
void deleteind (struct node * liste);
int fcheck (bool *r,int yer,int k);
float lcgrand(int stream);
void lcgrandst(long zset, int stream);
long lcgrandgt(int stream);
struct node * ReplaceWParentNoList(struct node *pop,double *fitness,int N, struct node *children,
double *childrenfitness, int chil, int * parentlist, int parentno,int k);
double MutateRandomOrt(struct node * individual, int derinlik, int N,double fitness);
struct node * seneratechildren (struct node * child,struct node * liste, int num,int N,bool * check);
struct llist * lrank (struct llist * rankl,double * fitness, int N,struct node * pop);
int * ranking(double *fitness, int * rank, int N);
struct node * ReplaceHalfParents(struct node *pop, double *fitness, int N, struct node *children,
double *childrenfitness, int chil, int * parentlist, int parentno,int k, int *parentrank, int *
childrenrank);
struct node * ReplaceAtLeastHalfParents(struct node *pop, double *fitness, int N, struct node
*children, double *childrenfitness, int chil, int * parentlist, int parentno,int k, int *parentrank, int *
childrenrank);
struct nodek * WriteToJList (struct nodek *liste,int parents, int count, int previous, int first);
struct nodek * CreateJUnion(struct node * child, int count,int N, struct nodek * liste);
struct node * GenerateJapaneseChildren (struct node * child, struct nodek * liste, int num,int N, int
parents,bool * check);
struct node * GenerateSJapaneseChildren (struct node * child, struct nodek * liste, int num,int N, int
parents,bool * check);
void assigne2(struct node * poop,struct node * chiil);
void edgebul(struct node * population, double * fintess, int N, int nodes, char * name);
void fenerate(struct node *bir, int N, bool * bak);
int kapasitele (int cap, int usage, int demand, int no);
int hop=0;
int repyap=0;
int longmutyap=0;
int randmutyap=0;
int longmutdene=0;
int randmutdene=0;
int capacity;

int randomseed=1;
int cggit=0,tccgit[3500];

double yol=50;
double dist[3500][3500];
int ortak[3500][3500];
int Ortakilk[3500];
int Ortakson[3500];
int demand[3500];
FILE * froute;
double prob[3500];
double avginit;

```

```

double VCost[10];
double Vwei[10];
int VCapacity[10];
int VNo[10];
double x[3500],y[3500];
int uzunluk, cno;
int seed;
char filew[20];
char filen[20];
double usunluk;
int onceki,usage;
char Vtipi[10];

int main(int argc, char **argv)
{
    int i,j,chil,de,
t,N,iterations,shortest,nodeci,termin,counter=1,bestcount=0,bestfind=0,souner=0,mut1=0,mut2=0,du
r=1,ilk;
    int repno;

    bool terminate = false;
    char filename[20];
    char fsol[20];
    char rfilename[20]="resa";
    char ek2[20]="m2a";
    char ek4[20]="m4a";
    char ek6[20]="m6a";
    char ek8[20]="m8a";
    char *nodenumber;
    FILE *fout;
    FILE *finput;
    FILE *foutput;
    FILE *frun2;
    FILE *frun4;
    FILE *frun6;
    FILE *frun8;
    int *parentlist;
    double *fitnss, *childrenfitnss, *hildrenfitnss, *pitnss,
mini,previousbest,etha,duration,worstninit=0,bestninit=0, deneme[3500],mutolasilik;
    int *ranked, *pranke, *cranke;
    bool * checker;
    struct node *iki;
    struct node *yaz;
    struct node * parents;
    struct node * children;
    struct node * uniongraph;
    struct nodek * kuniongraph;
    struct node * pop;
    struct node * sonucsirali;
    long double cpu_time_used;
    clock_t start,end;
    finput=fopen("input.txt", "r");
    Vtipi[0]='A';
    Vtipi[1]='B';
    Vtipi[2]='C';
    Vtipi[3]='D';
    Vtipi[4]='E';
    Vtipi[5]='F';
    Vtipi[6]='G';

```

```

Vtipi[7]='H';
Vtipi[8]='Y';
seed=atoi(argv[5]);
randomseed=1;
do
{
    etha=1;
    termin=1;
    iterations =10000;
for(i=0;i<20;i++)
filename[i] = argv[1][i];
for(i=0;i<20;i++)
fsol[i] = argv[2][i];
    for(i=0;i<20;i++)
filew[i] = argv[3][i];
    for(i=0;i<20;i++)
filel[i] = argv[4][i];

rfilename[0]='m';
rfilename[1]='e';
rfilename[2]='s';
rfilename[3]='a';
rfilename[4]=NULL;
strcat(rfilename,filename);
strcat(ek2,filename);
strcat(ek4,filename);
strcat(ek6,filename);
strcat(ek8,filename);

foutput=fopen(rfilename,"a");
k=nodeci;
k++;
nodeci++;
for(i=0;i<10;i++)
{
    Vwei[i]=1000;
    VNo[i]=100;
}

readdistances(filename);

k=uzunluk;
    N=k;
do
{
cr=2;
if(cr>N)
    printf("Number of parents cannot be higher than the population!\n");
}while(cr>N);

chil=1;
capacity = VCapacity[0];
pop = new struct node[N+1];
parents=new struct node[cr+1];
ranked=new int[N];
childrenfitnss=new double[chil+1];
hildrenfitnss=new double[chil+1];
parentlist=new int[cr+1];
children = new struct node[chil+1];

```

```

checker = new bool [k+3];
uniongraph=new struct node [k];
kuniongraph = new struct nodek [k];
fitnss=new double[N+3];

pranke=new int [cr+1];
cranke=new int [chil+1];
pitnss=new double [cr+1];
t=123;
for(i=0;i<cr;i++)
    parentlist[i]=0;
foutput=fopen(rfilename, "a");
if(randomseed==1)
{
fclose(foutput);
}
for(repno=1;repno<=dur;repno++)
{
    longmutyap=0;
    randmutyap=0;
    longmutdene=0;
    randmutdene=0;
    counter=0;
    bestcount=0;
    terminate=false;
    lcgrandst(seed, randomseed);
    froute=fopen(fsol, "r");
    for(i=0;i<N;i++)
    {
        generate(&pop[i],k,checker);
    }
    fclose(froute);
    repyap=0;
    cggit=0;
    uniongraph = createunion(parents,0,k, uniongraph);
    if(uzunluk>20)
    {
        yol=100;
    }
    fitnss=assignfitness(fitnss,pop,N);
    previousbest=fitnss[N+1];
    InitializeStat(parents, pop,cr-1,fitnss,parentlist,N);
    start=clock();
    printf("\nTUR %d\n",repno);
    while(!terminate)
    {
        counter++;

        previousbest=fitnss[N+1];
        selectrand(parents, pop,cr-1,fitnss,parentlist,N);
        if(counter==1)
        {
            avginit=fitnss[N];
            bestinit=fitnss[N+1];
            worstinit=fitnss[N+2];
        }

        for(i=0;i<cr;i++)
        {

```

```

        parents[i].plc=pop[parentlist[i]].plc;
        parents[i].nextnode=pop[parentlist[i]].nextnode;
        pitnss[i]=fitnss[parentlist[i]];

    }
    pranke=ranking(pitnss,pranke,cr);
    if(counter>10000)
    {
        printf("DUR");
        hop=1;
    }
    uniongraph = createunion(parents,cr,k, uniongraph);
    if(counter>10000)
listeyaz(uniongraph,k);
    if(counter==1)
        children=seneratechildren(children,uniongraph,chil,k,checker);
    else
    {
        if(counter>10000)
        {
            printf("parentlar\n");
            for(de=0;de<cr;de++)
            {
                printf("%lf ",parents[de]);
                yazdir(parents[de],k);
            }
            listeyaz(uniongraph,k);
        }

        children = generatechildren(children,uniongraph,chil,k,checker);
    }

    if(hop==1)
    {
        printf("childrenlar\n");
        for(de=0;de<chil;de++)
        {
            printf("%lf ",childrenfitnss[de]);
            yazdir(children[de],k);
        }
        childrenfitnss=assignfitness(childrenfitnss,children,chil);
        cranke=ranking(childrenfitnss,cranke,chil);
    for(i=0;i<chil;i++)
        childrenfitnss[cranke[i]]=MutateRandomOrt(&children[cranke[i]],1,k,childrenfitnss[cranke[
i]]);

        childrenfitnss=assignfitness(childrenfitnss,children,chil);
        if(childrenfitnss[0]==245)
        childrenfitnss=assignfitness(childrenfitnss,children,chil);

        pop=ReplaceWParentNoList(pop,fitnss,N,children,childrenfitnss,chil,parentlist,cr,k);

        if(previousbest==fitnss[N+1])
            bestcount++;
        else
        {

```

```

        sounter=bestcount;
        bestcount=0;
        bestfind=counter;
    }

    listesil(uniongraph,k);
    end=clock();
    duration=(end-start)/CLOCKS_PER_SEC;
    if((termin==1)&&(counter>=iterations))
        terminate=true;
    if((termin==2)&&(fitnss[N]==fitnss[N+1]))
        terminate=true;
    if((termin==3)&&(bestcount>=iterations))
        terminate=true;
    if((termin==4)&&((counter>=10000)||(fabs(fitnss[N+1]-
fitnss[N])<0.005)||(bestcount>=1000)))
        terminate=true;

    else
    {

        if(duration>3600)
        {
            terminate=true;
            printf("SURE BITTI");
            foutput=fopen(rfilename,"a");
            fprintf(foutput,"SURE BITTI");
            fclose(foutput);
        }
    }

    printf("bestcount %d , count %d",bestcount,counter);
    foutput=fopen(rfilename,"a");
    fprintf(foutput," %s, %d, %d, %.2lf, %.2lf, %d, %d, %.4lf, %4lf, %4lf, %4lf,
%.4lf, %.4lf, %d, %d, %d, %d, %d, %d, %d\n ",
        filename,cr,chil,          etha,
duration,counter,sounter,bestinit,avginit,worstinit,fitnss[N+1],fitnss[N],fitnss[N+2], repyap, bestfind,
cggit,longmutdene, longmutyap, randmutdene, randmutyap);
    fclose(foutput);

    printf("\n At time %lf, best is %lf, average is %lf, worst is %lf\n
",duration,fitnss[N+1],fitnss[N],fitnss[N+2]);

    mini = fitnss[1];
    shortest=1;
    for(i=1;i<N;i++)
    {
        if(fitnss[i]<mini)
        {
            shortest=i;
            mini=fitnss[shortest];
        }
    }
    printf("The shortest individual is %d with lenght %lf\n", shortest,mini);
    sonucsirali=&pop[shortest];
    frun2=fopen("Best_GA.txt","a");
    printf("Best\n");
    usunluk=0;

```

```

ilk =0;
onceki=0;
for(i=1;ilk<uzunluk;i++)
{
    printf("%d ",sonucsirali->plc);
    fprintf(frun2,"%d ",sonucsirali->plc);
    sonucsirali=sonucsirali->nextnode;

    usage=usage+demand[sonucsirali->plc];
    usunluk=usunluk+dist[onceki][sonucsirali->plc];
    onceki=sonucsirali->plc;
    if(sonucsirali->plc==0)
    {
        for(j=0;j<cno;j++)
            if(VCapacity[j]>usage)
                break;
        printf("\n Cost %lf, usage %d\n",VCost[j], usage);
        fprintf(frun2," : %d : %c :", usage,Vtipi[j]);
        usage=0;
    }
    else
        ilk++;
}
printf("%lf",usunluk);
printf("-1 ");
fprintf(frun2,"\n");
fclose(frun2);
for(i=0;i<N;i++)
{
    deleteind(&pop[i]);
}
}
printf("heyyeok");
randomseed--;

}while(randomseed>0);
fclose(foutput);
return 0;
}

```

```

void generate(struct node *bir, int N, bool * bak)
{
    struct node *sonraki,*emanet;
    int i,j;
    struct node *iki;
    int doluluk=0;
    for(i=0;i<N;i++)
        bak[i]=false;
    double num;
    sonraki=(struct node *)malloc(sizeof(node));
    bir->nextnode = sonraki;
    bir->plc = 0;
    capacity=VCapacity[0];
    for(i=0;i<N-1;i++)
    {
        sonraki->plc=fcheck(bak,i,N-1);
    }
}

```



```

        if(doluluk+demand[sonraki->plc]>capacity)
            capacity=VCapacity[kapasitele(capacity,
>plc],demand[sonraki->plc],0)];
        if(doluluk+demand[sonraki->plc]>capacity)
        {
            doluluk=0;
            iki=(struct node*)malloc(sizeof(node));
            iki->plc=sonraki->plc;
            sonraki->plc=0;
            sonraki->nextnode=iki;
        //    i--;
            sonraki=iki;
            doluluk=doluluk+demand[iki->plc];
        }
        else
            doluluk=doluluk+demand[sonraki->plc];

        if(i==k-2) break;
        emanet=(struct node*)malloc(sizeof(node));
        sonraki->nextnode=emanet;
        sonraki=emanet;
    }
    sonraki->nextnode=bir;
}

void fenerate(struct node *bir, int N, bool * bak)
{
    struct node *sonraki,*emanet;
    int i,j,ikinokta=0,nod;
    struct node *iki;
    char bos, * say;
    int doluluk=0;
    double num;
    say=new char [20];
    sonraki=(struct node *)malloc(sizeof(node));
    bir->nextnode = sonraki;
    capacity=VCapacity[0];
    for(i=0;i<6*N-1;i++)
    {
        if(i==0)
            bir->plc=nod;
        else
            sonraki->plc=nod;
        fscanf(frout,"%d",&nod);
        if(nod<0)
            break;
        if(i!=0)
        {
            emanet=(struct node*)malloc(sizeof(node));
            sonraki->nextnode=emanet;
            sonraki=emanet;
        }
    }
    sonraki->nextnode=bir;
}

```

int fcheck (bool *r,int yer,int k) // this fuunction uses a boolean list, too keep and check if the node was previously visited, if visited the value is made true and this node is not assigned any mode

```
{
    int i,count=0,j=0,sayi,tut;
    bool search=false;

    do
    {
        tut=float(lgrand(randomseed)*(k-yer));
        sayi = tut+1;

        for(i=0;i<=k;i++)
        {
            if(r[i]==false) count++;

            if (count==sayi)
            {
                r[i]=true;
                j=i;
                search=true;
                break;
            }
        }

    }while (search==false);

    return j+1;
}
```

```
void readdistances(char * filename)
{
    double result,hold;
    int i,j,k,max;
    FILE *fread;
    FILE *fnum;
    FILE *fwei;
    fread = fopen(filename,"r");
    printf("dosyayy actim, %s\n",filename);
    fscanf(fread,"%d %d",&uzunluk, &cno);

    if(uzunluk>30)
    {
        fnum=fopen(filen,"r");
        fwei=fopen(filew,"r");
        fscanf(fwei,"%d",&max);
    }
    for(i=0;i<cno;i++)
    {
        fscanf(fread,"%d",&VCapacity[i]);
        fscanf(fread,"%lf",&VCost[i]);
        if((uzunluk>30)&&(i<max))
        {
            fscanf(fnum,"%d",&VNo[i]);
            fscanf(fwei,"%lf",&Vwei[i]);
        }
        else
        {
```

```

        VNo[i]=100;
        Vwei[i]=1;
    }

}
for(i=0;i<uzunluk;i++)
{
    fscanf(fread,"%d %lf %lf\n",&k,&x[i],&y[i]);
}
for(i=0;i<uzunluk;i++)
{
    fscanf(fread," %d %d\n",&k,&demand[i]);
}
fclose(fread);
for(i=0;i<uzunluk;i++)
    for(j=0;j<uzunluk;j++)
    {
        if(i==j)
            dist[i][j]=0;
        else
        {
            hold=sqrt(pow((x[i]-x[j]),2) + pow((y[i]-y[j]),2));
            if(hold*1000000-floor(hold*1000000)<0.5)
                dist[i][j]=floor(hold*1000000)/1000000;
            else dist[i][j]=floor((hold+1)*1000000)/1000000;
        }
    }
}

void leaddistances(char * filename, int N)
{
    double x[3500],y[3500],result,hold;
    int i,j,k;
    FILE *fread;
    fread = fopen(filename,"r");
    printf("dosyayı actım, %s\n",filename);
    for(i=0;i<N;i++)
    {
        fscanf(fread," %d %lf %lf %d\n",&k,&x[i],&y[i],&demand[i]);
        printf("%d %lf %lf %d \n",i,x[i],y[i], demand[i]);
    }
    fclose(fread);
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        {
            if(i==j)
                dist[i][j]=0;
            else
            {
                hold=sqrt(pow((x[i]-x[j]),2) + pow((y[i]-y[j]),2));
                if(hold-floor(hold)<0.5)
                    dist[i][j]=floor(hold);
                else dist[i][j]=floor(hold+1);
            }
        }
}

```

```

}

double * assignfitness (double * fitnss, struct node * input,int length)
{
    int first=NULL,previous=-1,current=0, usage=0, il,im, Hno[10];
    struct node * hold,*told;
    int i,j;
    double sum=0,tum;
    tum=0;
    hold=input[0].nextnode;
    first=hold->plc;
    hold=hold->nextnode;
    for(i=0;i<cno;i++)
    {
        Hno[i]=0;
    }

    for(i=0;i<uzunluk*2;i++)
        if((hold->plc!=0))
        {
            hold=hold->nextnode;
        }
        else
        {
            if (hold->nextnode->plc!=0)

                break;
        }

    hold=hold->nextnode;
    first=hold->plc;
    previous=first;
    hold=hold->nextnode;
    usage=demand[first];
    sum=sum+dist[0][previous];
    for(i=1;i<length+1;i++)
    {
        do
        {

            sum = sum + dist[hold->plc][previous];
            previous=hold->plc;
            hold=hold->nextnode;

            if (previous==0)
            {
                for(il=0;il<cno;il++)
                {
                    if(VCapacity[il]>=usage)
                        break;
                }
                if(Hno[il]<VNo[il])
                {
                    Hno[il]++;

                    tum=tum+Vwei[il]*sum+ VCost[il];
                    sum=0;
                }
            }
        }
    }
}

```

```

    }
    else if (uzunluk>30)
        tum=tum+sum*100+1000000;
    }
    if(previous==0)
    {
        usage=0;
    }
    else
        usage=usage+demand[previous];

}while(hold->plc != first);

    fitnss[i-1]=tum;
    sum=0;
    tum=0;
    if(i==length)
        break;
    hold=input[i].nextnode;
for(im=0;im<uzunluk*2;im++)
if((hold->plc!=0))
{
    hold=hold->nextnode;
}
else
{
    if (hold->nextnode->plc!=0)

        break;
}

hold=hold->nextnode;
first=hold->plc;
for(il=0;il<cno;il++)
{
    Hno[il]=0;
}

for(il=0;il<length+1;il++)
if(hold->plc!=0)
    hold=hold->nextnode;
else break;

hold=hold->nextnode;
first=hold->plc;
previous=first;
hold=hold->nextnode;
usage=demand[first];
sum=sum+dist[first][previous];
}
return fitnss;
}

```

```

void InitializeStat(struct node * one, struct node * input,int count,double *fitnss,int * parents,int N)
{
    int i,j;
    double mini,maxi;
    double average=0;
    struct node hold;
    mini = fitnss[0];
    maxi = fitnss[0];
    j=0;
    for(i=1;i<N;i++)
    {
        if(fitnss[i]<mini)
        {
            j=i;
            mini=fitnss[j];
        }
        average=average+fitnss[i];
        if(fitnss[i]>maxi)
        {
            j=i;
            maxi=fitnss[j];
        }
    }
    fitnss[N+1]=mini;
    fitnss[N+2]=maxi;
    fitnss[N]=average/double(N-1);
}

void selectrand(struct node * one, struct node * input,int count,double *fitnss,int * parents,int N)
{
    int i,j,m;

    struct node hold;
    for(i=0;i<=count;i++)
    {
        j=float(lcgrand(randomseed)*(N-i));

        for(m=0;m<i;m++)
            if(j>=parents[m]) j++;
        parents[i]=j;
        one[i]=input[j];
    }
}

struct node * createunion(struct node * child, int count,int N, struct node * liste)
{
    int i,j,veryfist,previous,first;
    struct node *hold;
    for(i=0;i<N;i++)
    {
        liste[i].plc=-1;
        liste[i].nextnode=NULL;
    }
    for(i=0;i<count;i++)
    {
        previous=child[i].nextnode->plc;
        veryfist=child[i].nextnode->plc;
    }
}

```

```

        hold=&child[i];
        hold=hold->nextnode;
        do
        {
            hold=hold->nextnode;
            first=hold->plc;
            if(first!=previous)
            liste=writetolist(liste,count,previous+1,first+1);
            if(hop==1)
            {
                printf(" %d %d %d\n",count,previous,first);
            }
            previous=first;
        }while(veryfist!=previous);
    }
    return liste;
}

```

```

struct node * writetolist (struct node *liste, int count, int previous, int first)

```

```

{
    int i,j,temp=-1,one,other;
    bool flag =false;
    struct node *hold,*sonraki;
    one=previous-1;
    other=first-1;

    for(i=0;i<2;i++)
    {

        hold=&liste[one];
        while (!flag){
            if((hold->nextnode==NULL) && (hold->plc==-1))
            {
                liste[one].plc=other;
                liste[one].nextnode=NULL;
                flag = true;
            }
            else
            {
                if(hold->nextnode==NULL)
                {
                    if(hold->plc==other) break;
                    if(dist[one][hold->plc]>dist[one][other])
                    {
                        temp=hold->plc;
                        hold->plc=other;
                        other=temp;
                    }

                    sonraki=(struct node *)malloc(sizeof(node));
                    hold->nextnode=sonraki;
                    sonraki->plc=other;
                    sonraki->nextnode=NULL;
                    flag=true;
                }
                else
                {
                    temp=hold->plc;

```

```

        if(temp==other) break;
        if(dist[one][temp]>dist[one][other])
        {
            sonraki=(struct node *)malloc(sizeof(node));
            sonraki->plc=hold->plc;
            hold->plc=other;
            sonraki->nextnode=hold->nextnode;
            hold->nextnode=sonraki;
            flag=true;
        }hold=hold->nextnode;
    }
    }}
    one=first-1;
    other=previous-1;
    hold=&liste[one];
    flag=false;
}
return liste;
}

struct node * seneratechildren (struct node * child,struct node * liste, int num,int N,bool * check)
{
    int i,j,mi,t,veryfirst,count, o,hasan;
    // tccgit=0;
    bool flag=false,change=false;
    //bool * check;
    double min,hold;
    struct node * me, *temp,*zonk,*init;
    int doluluk=0,hlag,old;
    // check = new bool[N+2];
    capacity=VCapacity[0];
    for(t=0;t<num;t++)
    {
        tccgit[t]=0;
        flag=false;
        check[0]=false;
        for(i=1;i<N+1;i++)
            check[i]=false;

        me=&child[t];
        init=me;
        if(t==0)
            me->plc=0;
        else
            me->plc=hasan;
        min=0;
        count=0;
        hold=0;
        zonk=&liste[me->plc];
        doluluk=doluluk+demand[me->plc];

        while(count<N-1)
        {
            hold=0;
            hlag=1;

            if(doluluk+demand[zonk->plc]>capacity)
                capacity=VCapacity[kapasitele(capacity, doluluk+demand[zonk-
                >plc],demand[zonk->plc],lcgrand(randomseed)*3)];

```



```

        if((check[zonk->plc] == false) && (doluluk+demand[zonk-
>plc]<=capacity)&&(zonk->plc!=old))
        {
            temp=(struct node*)malloc(sizeof(struct node));
            temp->plc=zonk->plc;
            temp->nextnode=NULL;
            me->nextnode=temp;
            old=me->plc;

            me=me->nextnode;
            if ( me->plc ==0)
            {
                doluluk=0;
            }
            else
                count=count+1;
            zonk=&liste[me->plc];
            if(me->plc!=0)
            check[me->plc]=true;
            change=true;

            doluluk= demand[me->plc]+doluluk;

        }
        else
        {

            if (zonk->nextnode==NULL)
            {
                flag = true;

            }
            else
            {
                zonk=zonk->nextnode;
            }
        }
        if(flag==true)
        {
            j=0;
            for(i=0;i<N;i++)
            {
                if((check[i]==false) && (doluluk+demand[i]<capacity) && (me-
>plc!=i))
                {
                    hlag=0;

                    if(hold==0)
                    {
                        j=i;
                        hold = dist[me->plc][i];
                        min=hold;
                        tccgit[t]++;
                    }
                }
            }
        }
    }
}

```

```

        hold = dist[me->plc][i];

        if((min>hold) && (hold!=0))
        {
            j=i;
            min=hold;
            tccgit[t]++;
        }
        else hlag*1;
    }
    hold=0;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->plc=j;
    temp->nextnode=NULL;
    me->nextnode=temp;

    me=me->nextnode;
    if(j==0)
    {

        doluluk=0;
    }
    else
        count=count+1;
    zonk=&liste[me->plc];
    check[me->plc]=true;

    doluluk=doluluk+demand[j];
    j=0;
    flag=false;
}
if(((hlag==0) && (flag==1))|| (doluluk==capacity))&& (count!=N-1)
{
    temp=(struct node*)malloc(sizeof(struct node));
    temp->plc=0;
    temp->nextnode=NULL;
    me->nextnode=temp;
    me=me->nextnode;
    zonk=&liste[me->plc];
    doluluk=0;
}
}
    hasan=me->plc;
    me->nextnode=&child[t];
}
    return child;
}

struct node * ReplaceWParentNoList(struct node *pop,double *fitness,int N, struct node *children,
double *childrenfitness, int chil, int * parentlist, int parentno,int k)
{
    int i,j,m,c,flag;
    double min,pax,old;

```

```

struct node * yoket, *temp;
struct llist *rank, * ara;
min=childrenfitness[0];
j=0;
for(i=1;i<chil;i++)
{
    if(min>childrenfitness[i])
    {
        min=childrenfitness[i];
        j=i;}
}

pax=fitness[parentlist[0]];
c=parentlist[0];
for(i=1;i<parentno;i++)
{

    if(pax<fitness[parentlist[i]])
    {
        pax=fitness[parentlist[i]];
        c=parentlist[i];
    }
}
old=fitness[c];
if(pax>min)
{
    fitness[c]=min;
    assigne2(&pop[c],&children[j]);
    repyap++;
    cggit=tccgit[j]+cggit;
    temp=&children[j];
if(fitness[c]<fitness[N+1])
    fitness[N+1] = fitness[c];
fitness[N]=fitness[N]+fitness[c]/N-old/N;
if(fitness[N+2]==old)
{
    fitness[N+2]=fitness[0];
    for(i=1;i<N;i++)
        if(fitness[N+2]<fitness[i])
        {
            fitness[N+2]=fitness[i];
        }
}
}
return pop;
}

```

```

void deleteind (struct node * liste)
{
    int i,j=0,first;
    struct node *uur,*rafet;
    if(liste->nextnode==NULL || liste->plc==0) return;
    first=liste->plc;
    rafet=liste;
    uur=rafet->nextnode;
    while(rafet->nextnode->plc!=first)
    {
        rafet=rafet->nextnode;
    }
    rafet->nextnode=NULL;
}

```

```

rafet=liste;

while(rafet!=NULL)
{
    j++;
    uur=rafet;
    rafet=rafet->nextnode;
    uur->nextnode=NULL;
    if(j>1)
    {
        free(uur);
    }
}
free(rafet);

liste->nextnode=NULL;
liste->plc=0;
}
void listesil (struct node * girilen, int N)
{
    int i,j=0;
    struct node * huhu, *tut;
    for(i=0;i<N;i++)
    {
        huhu=&girilen[i];
        j=0;
        while (1)
        {
            j++;
            tut=huhu;
            if(huhu==NULL)
            {
                break;
            }
            else
            {
                huhu = huhu->nextnode;
                if(j>1)
                free(tut);
                tut=NULL;
            }
        }
    }
}

struct node assigne(struct node * poop,struct node * chiil)
{
    int i,j,t,first,count=0;
    struct node * temp,fist,*hold;
    first=chiil->plc;
    hold=poop;
    fist.plc=chiil->plc;
    chiil=chiil->nextnode;
    poop->plc=chiil->plc;
    poop->nextnode=NULL;
    chiil=chiil->nextnode;
    do
    {

```

```

        count++;
        temp=(struct node*)malloc(sizeof(struct node));
        temp->plc=chiil->plc;
        temp->nextnode=NULL;
        if (count==1)
            fist.nextnode=poop;
        poop->nextnode=temp;
        poop=poop->nextnode;
        chiil=chiil->nextnode;
    }while(chiil->plc!=first);
    poop->nextnode=hold;
    return fist;
}

static long zrng[] =
{
    1,
    1973272912, 281629770, 20006270,1280689831,2096730329,1933576050,
    913566091, 246780520,1363774876, 604901985,1511192140,1259851944,
    824064364, 150493284, 242708531, 75253171,1964472944,1202299975,
    233217322,1911216000, 726370533, 403498145, 993232223,1103205531,
    762430696,1922803170,1385516923, 76271663, 413682397, 726466604,
    336157058,1432650381,1120463904, 595778810, 877722890,1046574445,
    68911991,2088367019, 748545416, 622401386,2122378830, 640690903,
    1774806513,2132545692,2079249579, 78130110, 852776735,1187867272,
    1351423507,1645973084,1997049139, 922510944,2045512870, 898585771,
    243649545,1004818771, 773686062, 403188473, 372279877,1901633463,
    498067494,2087759558, 493157915, 597104727,1530940798,1814496276,
    536444882,1663153658, 855503735, 67784357,1432404475, 619691088,
    119025595, 880802310, 176192644,1116780070, 277854671,1366580350,
    1142483975,2026948561,1053920743, 786262391,1792203830,1494667770,
    1923011392,1433700034,1244184613,1147297105, 539712780,1545929719,
    190641742,1645390429, 264907697, 620389253,1502074852, 927711160,
    364849192,2049576050, 638580085, 547070247 };

float lcgrand(int stream)
{
    long zi, lowprd, hi31;

    zi = zrng[stream];
    lowprd = (zi & 65535) * MULT1;
    hi31 = (zi >> 16) * MULT1 + (lowprd >> 16);
    zi = ((lowprd & 65535) - MODLUS) +
        ((hi31 & 32767) << 16) + (hi31 >> 15);
    if (zi < 0) zi += MODLUS;
    lowprd = (zi & 65535) * MULT2;
    hi31 = (zi >> 16) * MULT2 + (lowprd >> 16);
    zi = ((lowprd & 65535) - MODLUS) +
        ((hi31 & 32767) << 16) + (hi31 >> 15);
    if (zi < 0) zi += MODLUS;
    zrng[stream] = zi;
    return (zi >> 7 | 1) / 16777216.0;
}

void lcgrandst (long zset, int stream) /* Set the current zrng for stream
                                     "stream" to zset. */
{
    zrng[stream] = zset;
}

```

```

long lcgrandgt (int stream) /* Return the current zrng for stream "stream". */
{
    return zrng[stream];
}

```

```

int * ranking(double *fitness, int * rank, int N)
{
    int i,j,ka;
    int tut,temp;
    double current;

    for(i=0;i<10;i++)
        rank[i]=0;

    rank[0]=0;
    for(i=1;i<N;i++)
    {
        temp=i;
        for(j=0;j<i;j++)
        {
            if(fitness[temp]<fitness[rank[j]])
            {
                tut=rank[j];
                rank[j]=temp;
                temp=tut;
            }
        }
        rank[i]=temp;
    }

    return rank;
}

```

```

void listeksil (struct nodek * girilen, int N, int parents)
{
    int i,j=0 ;
    struct nodek * huhu, *tut;
    for(i=0;i<N;i++)
    {
        huhu=&girilen[i];
        j=0;
        while (1)
        {
            j++;
            tut=huhu;
            if(huhu==NULL)
            {
                break;
            }
            else
            {
                delete(tut->parents);
                huhu = huhu->nextnode;
                if(j>1)
                    free(tut);
            }
        }
    }
}

```

```

        tut=NULL;
    }
}

double MutateRandomOrt(struct node * individual, int derinlik, int N,double fitness)
{
    int yer,i,j,gum,flag=0,ilk,tum=0,il;
    double newfitness,saving,loss, deger=-1, cumulatif=0, doluluk=0,a,b;
    struct node * burasi, *omer, *cikart, *tak, *cak, *tas,*temp;
    omer=individual;
    for(gum=1;gum<=derinlik;gum++)
    {
        yer=float(lcgrand(randomseed)*N)+1;
        burasi=individual;
        for(i=0;i<yer;i++)
        {
            if(burasi->plc==0)
                doluluk=0;
            else
                doluluk=doluluk+demand[burasi->plc];

            burasi=burasi->nextnode;
            if(burasi->plc==0)
                i--;
        }
        tas=burasi;
        for(i=0;i<N;i++)
        {
            if(tas->plc==0)
                break;
            if(tas->plc!=burasi->nextnode->plc)
                doluluk=doluluk+demand[burasi->plc];
            tas=tas->nextnode;
        }
        if(burasi->nextnode->plc==0)
        {
            burasi=burasi->nextnode;

            doluluk=0;
            tas=burasi->nextnode;

            for(i=0;i<N;i++)
            {
                if(tas->plc==0)
                    break;
                if(tas->plc!=burasi->nextnode->plc)
                    doluluk=doluluk+demand[tas->plc];
                tas=tas->nextnode;
            }
        }
        saving=-dist[burasi->plc][burasi->nextnode->nextnode->plc]+(dist[burasi->plc][burasi->nextnode->plc]+dist[burasi->nextnode->plc][burasi->nextnode->nextnode->plc]);
        omer=burasi->nextnode->nextnode;
        cak=burasi;
    }
}

```

```

burasi=burasi->nextnode;

for(j=0;j<N && omer->nextnode->nextnode != burasi;j++)
{
    if(omer->plc==0)
    {
        tas=omer->nextnode;
        doluluk=0;
        for(i=0;i<N;i++)
        {
            if(tas->plc==0)
                break;
            if(tas->plc!=burasi->nextnode->plc)
                doluluk=doluluk+demand[tas->plc];
            tas=tas->nextnode;
        }
    }
    loss=dist[omer->plc][burasi->plc]+dist[burasi->plc][omer->nextnode->plc]-dist[omer->plc][omer->nextnode->plc];
    if((deger== -1) && (saving>loss))
        deger=saving-loss;
    for(il=0;il<cno;il++)
    {
        if(VCapacity[il]>=tum)
            break;
    }
    capacity = VCapacity[il];
    if(((saving-loss)>=deger) && (demand[burasi->plc]+doluluk<capacity))
    {
        flag=1;
        cikart=cak;
        tak=omer;
        deger=saving-loss;
    }
    omer=omer->nextnode;
}

if(flag==1)
{
    omer=tak->nextnode;
    tak->nextnode=cikart->nextnode;
    cikart->nextnode=cikart->nextnode->nextnode;
    tak->nextnode->nextnode=omer;
    cumulatif=cumulatif+deger;
    flag=0;
    randmutyap++;
}
}
burasi=individual->nextnode;
if(burasi->plc==0)
{
    individual->nextnode=individual->nextnode->nextnode;
    free(burasi);
}
burasi=individual->nextnode;
for(i=0;i<N*2;i++)
{
    if((burasi->plc==burasi->nextnode->plc))
    {

```



```

        burasi->nextnode=burasi->nextnode->nextnode;
    }
    else if(burasi->nextnode->plc==individual->nextnode->plc)
        break;
    burasi=burasi->nextnode;
}
randmutdene=randmutdene+gum-1;
return fitness-cumulatif;
}

int kapasitele (int cap, int usage, int demand, int no)
{
    int i,j,c;
    int tip, nip, ilk;
    for(i=0; VCapacity[i+1]<=cap; i++);
    tip=i;
    if(no==0)
    {
        nip=tip + lcgrand(randomseed)*(cno-tip);
    }
    else if(no==1)
    {
        nip=-1;
        ilk=-1;
        for(c=0;c<cno;c++)
        {
            if((usage<VCapacity[c] && (nip<0))
                nip=c;
            if((demand<VCapacity[c]&&(ilk<0))
                ilk=c;
            if((ilk>=0)&&(nip>=0))
                break;
        }
        if(!((ilk>=0)&&(VCost[tip]+VCost[ilk]>VCost[nip])) && (nip>=0))
        {
            nip=tip;
        }
    }
    else
    {
        nip=-1;
        ilk=-1;
        for(c=0;c<cno;c++)
        {
            if((usage<VCapacity[c] && (nip<0))
                nip=c;
            if((demand<VCapacity[c]&&(ilk<0))
                ilk=c;
            if((ilk>=0)&&(nip>=0))
                break;
        }
        if(!((ilk>=0)&&(VCost[tip]*(VCapacity[tip]-(usage-
demand))+VCost[ilk]*(VCapacity[ilk]-demand))>(VCost[nip]*(VCapacity[nip]-usage)) &&
(nip>=0)))
        {
            nip=tip;
        }
    }
}

```

```

        return nip;
    }

void assigne2(struct node * poop,struct node * chiil)
{
    int tut,i,j,cilk,pilk,stop1=0, stop2=0;
    struct node * hah, * first, * tirst, * tah, * vah;
    first=chiil;
    tirst=poop;

    pilk=poop->nextnode->plc;
    cilk=chiil->nextnode->plc;
    poop->plc=chiil->plc;
    hah=poop->nextnode;
    tah=hah;
    vah=chiil->nextnode;

    for(i=0;i<k*2;i++)
    {
        if(chiil->nextnode->nextnode->plc==cilk)
        {
            chiil->nextnode=poop;
            stop1=1;
        }
        if(hah->nextnode->nextnode->plc==pilk)
        {
            hah->nextnode=first;
            stop2=1;
        }
        if((stop1==0)||(stop2==0))
        {
            if(stop1!=1)
                chiil=chiil->nextnode;
            if(stop2!=1)
                hah=hah->nextnode;
        }
        else
            break;
    }
    first->nextnode=tah;
    tirst->nextnode=vah;
}

struct node * generatechildren (struct node * child,struct node * liste, int num,int N,bool * check)
{
    int i,j,mi,t,veryfirst,count, o,hasan,kac,bas;
    bool flag=false,change=false,one=false;
    double min,hold;
    struct node * me, * temp,*zonk,*init,*hemp,*tas;
    int doluluk=0,hlag,old;
    capacity=VCapacity[0];
    for(t=0;t<num;t++)
    {
        tccgit[t]=0;
        flag=false;
        check[0]=false;
        for(i=1;i<N+1;i++)
            check[i]=false;
        kac=child[t].nextnode->plc;
    }
}

```

```

temp=child[t].nextnode;
for(i=0;i<uzunluk*2;i++)
{
    if(temp->nextnode->plc==kac)
    {
        temp->plc=-1;
        break;
    }
    else
        temp=temp->nextnode;
}

me=&child[t];
init=me;
child[t].plc=-1;
me->plc=0;
min=0;
count=0;
hold=0;
doluluk=doluluk+demand[me->plc];
me->plc=-1;
me=me->nextnode;
bas=zonk->plc;
while(count<N-1)
{
    hold=0;
    hlag=1;
    if(doluluk+demand[zonk->plc]>capacity)
        capacity=VCapacity[kapasitele(capacity, doluluk+demand[zonk-
>plc],demand[zonk->plc],lcgrand(randomseed)*3)];
    if((check[zonk->plc] == false) && (doluluk+demand[zonk-
>plc]<=capacity)&&(zonk->plc!=old))
    {
        me->plc=zonk->plc;
        old=me->plc;
        if ( me->plc == 0)
        {
            doluluk=0;
        }
        else
            count=count+1;
        if(one==false)
            bas=me->plc;
        one=true;
        if((me->nextnode->plc==-1)&&(count<N-1))
        {
            temp=(struct node*)malloc(sizeof(struct node));
            temp->nextnode=me->nextnode;
            me->nextnode=temp;
            temp->plc=0;
        }
        if((count==N-1))
            break;
        me=me->nextnode;
    }
}

```

```

        zonk=&liste[old];
        if(old!=0)
        check[old]=true;
        change=true;
        doluluk= demand[old]+doluluk;

    }
    else
    {

        if (zonk->nextnode==NULL)
        {
            flag = true;

        }
        else
        {
            zonk=zonk->nextnode;
        }
    }
    if(flag==true)
    {
        j=0;
        for(i=1;i<N;i++)
        {
            if((doluluk+demand[i]>capacity)           &&
                (check[i]==false))
            {
                for(mi=0;mi<cno;mi++)
                {
                    if(demand[i]<capacity)
                        break;
                    else

                }

                capacity=VCapacity[mi];

                capacity=VCapacity[kapasitele(capacity,
                doluluk+demand[i],demand[i],lcgrand(randomseed)*3)];
            }
            if((check[i]==false) && (doluluk+demand[i]<capacity)
                && (old!=i))
            {
                hlag=0;

                if(hold==0)
                {
                    j=i;
                    hold = dist[old][i];
                    min=hold;
                    tccgit[t]++;
                }
                hold = dist[old][i];

                if((min>hold) && (hold!=0))
                {
                    j=i;

```

```

        min=hold;
        tccgit[t]++;
    }
}
else hlag*1;

}
hold=0;
me->plc=j;
old=j;
if(j==0)
{
    doluluk=0;
}
else
    count=count+1;

if((me->nextnode->plc==-1)&&(count<N-1))
{
temp=(struct node*)malloc(sizeof(struct node));
temp->nextnode=me->nextnode;
me->nextnode=temp;
temp->plc=0;
}
if((count==N-1))
break;
me=me->nextnode;
zonk=&liste[old];
if(old!=0)
check[old]=true;
change=true;
doluluk=doluluk+demand[old];
j=0;
flag=false;
}
if((((hlag==0) && (flag==1))|| (doluluk==capacity))&& (count!=N-1))
{
me->plc=0;
if((me->nextnode->plc==-1)&&(count<N-1))
{
temp=(struct node*)malloc(sizeof(struct node));
temp->plc=0;
temp->nextnode=NULL;
temp->nextnode=me->nextnode;
me->nextnode=temp;
}
else
me=me->nextnode;
zonk=&liste[0];
doluluk=0;
}
}
hasan=me->plc;
if((me->plc!=-1) && (me->nextnode!=NULL))
{
temp=me->nextnode;
for(i=0;i<uzunluk;i++)

```

```

        {
            if((temp->plc==-1)&&(temp->nextnode->plc==bas))
            {
                me->nextnode=temp;
                break;
            }
            else
            {
                hemp=temp;
                temp=temp->nextnode;
                hemp->nextnode=NULL;
                free(hemp);
            }
        }
    }
    child[t].plc=0;

    temp=child[t].nextnode;
    for(i=0;i<uzunluk*2;i++)
    {
        if(temp->plc==-1)
        {
            temp->plc=0;
            break;
        }
        else
            temp=temp->nextnode;
    }
    temp=&child[0];
    for(i=0;i<2*N;i++)
    {
        if(temp->plc!=0)
            break;
        else
            temp=temp->nextnode;
    }
    j=temp->plc;
    temp=temp->nextnode;
    for(i=0;i<2*N;i++)
    {
        if((temp->plc==0)&&(temp->nextnode->plc==0))
        {
            tas=temp->nextnode;
            temp->nextnode=temp->nextnode->nextnode;
            tas->nextnode=NULL;
            free(tas);
        }
        else
            temp=temp->nextnode;
        if(j==temp->plc)
            break;
    }
    if(hop==1)
    {
        printf("Childi yazdiriyorum!!\n");
        yazdir(child[t],70);
    }
}return child;}

```

APPENDIX F

The C codes of the multi-agent system are also provided in the Appendix CD, attached on the back cover of the thesis.

```
#include<iostream.h>
#include<iomanip.h>
#include<stdlib.h>
#include<time.h>
#include<fstream>
#include<math.h>
#include<string.h>
#include<time.h>

void readdistances(char * filename);
void teaddistances(char * filename);
int fcheck (bool *r,int k);
void initialize ();
void yazdir(int sayi);
void findandbbwrite(int ajannono);
void giveaprice(int ajanno);
void sendToCentralAgent(void);
void selectbest(void);
void update(int cityno, int verenajan, int alanajan, int konacakyer);
void improve(int i);
void hepsiniduyur(int ajanno);
void selectranking(double etha);
void find2write(int ajanno);
void givesprice(int ajanno);
void findandwrite(int ajanno);
{
    int no;
    int cap;
};

struct boardnode
{
    int cap;
    double savings;
    int ownerid;
    int bas;
    int son;
    int sayi;
};

struct saver {
    double savings;
    int place;
    int yon;
};
```

```

double dist[300][300];
int capacity[300];
double x[500],y[500];
struct node ajan[300][300];
int nodesayisi;
int aracsayisi;
struct boardnode blackb[300];
struct saver keep[300][300];
int vehiclecapacity[300];
int dcap[300];
double dcost[300];
int dahaoncevermistim[300];
int tradeyaptim[300];
int doluluk[300];
int run;
int improveyap=0;
double selector=1;
int method;
double alpha[300];
int kandir=0;
int initializer=0;
int conv=0;

char filew[20];
int main(int argc, char **argv)
{
    int i,j,k,bu,su,count,tradeyapilmis;
    int repno,replication,runlength=10,hepsi=1;
    double tum=0,zost=0;
    char filename[20];
    for(i=0;i<20;i++)
    {
        filename[i]=argv[1][i];
    }

    for(i=0;i<20;i++)
    {
        filew[i]=argv[2][i];
    }

    char *nodenumber;
    FILE *fout;
    FILE *fname;
    FILE *finput;
    FILE *foutput;
    long double cpu_time_used;
    double sum;
    clock_t start,end;
    do{
        repno=10;
        runlength=-1;
        improveyap=1;
        selector=atof(argv[5]);
        method=atoi(argv[4]);
        kandir=atoi(argv[3]);
        if(repno <=replication ) break;
        readdistances(filename);
    }for(replication=0;replication<repno;replication++)
    {

```



```

runlength=300;
sum=0;
zost=0;
conv=0;
for(i=0;i<nodesayisi+2;i++)
{
    for(j=0;j<nodesayisi+2;j++)
    {
        ajan[i][j].no=0;
        ajan[i][j].cap=0;
        keep[i][j].savings=-1;
        keep[i][j].place=-1;
    }
    blackb[i].ownerid=0;
    blackb[i].savings=0;
    vehiclecapacity[i]=dcap[aracsayisi-1];
    dahaoncevermistim[i]=0;
}
initialize();
for(run=0;((run<runlength) || (runlength ==-1));run++)
{
    for(i=0;i<=nodesayisi+1;i++)
        if(ajan[i][0].cap!=0)
        {
            if(kandir==1)
                findandwrite(i);
            else
            {
                if(method==1) findandbbwrite(i);
                else if(method==2) hepsiniduyur(i);
                else find2write(i);
            }
        }
}
for(i=0;i<=nodesayisi+1;i++)
    if(ajan[i][0].cap!=0)
    {
        if(kandir==1)
            givesprice(i);
        else
            giveaprice(i);
    }
hepsi=1;
for(i=0;i<nodesayisi;i++)
{
    for(j=0;j<=nodesayisi;j++)
    {
        if(keep[i][j].savings>0)
        {
            hepsi=0;
        }
    }
}
sendToCentralAgent();
tradeyapilmis=0;
for(i=0;i<nodesayisi+2;i++)
    if(tradeyaptim[i]!=0)

```

```

                                tradeyapilmis=1;
if(tradeyapilmis==0)
{
    runlenght=0;
    conv++;
}
else conv=0;

for(i=0;i<nodesayisi+2;i++)
{
    tradeyaptim[i]=0;
}
if(improveyap>0)
for(i=0;i<nodesayisi;i++)
    improve(i);
if((runlenght==1) && (hepsi==1) || (run>=50))
    runlenght=0;
}

sum=0;
foutput=fopen("anyresults.txt","a");
fout=fopen("anydis_res.txt","a");

count=0;
fprintf(foutput,"File %s rep %d (init %d / method %d / kandir %d / select %lf / imp. %d:
\n", filename,replication, initializor, method, kandir,selector, improveyap);
for(i=0;i<nodesayisi+1;i++)
{
    if(ajan[i][1].no!=0)
    {
        count++;
        fprintf(foutput,"Ajan %d: ", count);
        for(j=0;j<nodesayisi+1;j++)
        {
            fprintf(foutput,"%d ",ajan[i][j]);
            su=ajan[i][j].no;
            if(j>=1)
                tum=dist[bu][su]+tum;
            bu=su;
            if(ajan[i][j].no==0 && j>0)
            {
                break;
            }
        }
        fprintf(foutput,"\n");
        for(k=0;dcap[k]>=0;k++)
        {
            if(dcap[k]>=ajan[i][0].cap)
            {
                vehiclecapacity[i]=dcap[k];
                zost=dcost[k];
                break;
            }
        }
        sum=sum+tum*alpha[k]+(double)zost;
    }
}

```

```

        tum=0;
    }

}
fprintf(fout,"FILE %s rep %d (%d %d %d %lf %d) %lf %d
\n",filename,replication,initializor, method, kendir, selector,improveyap,sum,count);
printf("%s (%d %d %lf) %lf %d\n",filename,method, kendir, selector,sum,count);
fclose(foutput);
fclose(fout);
}
}
while(true);
return 0;

}

void teaddistances(char * filename)
{
    double x[500],y[500],result,hold;
    int i,j,k,h;
    FILE *fread;
    FILE * dista;

    fread = fopen(filename,"r");

    printf("dosyayı actim, %s\n",filename);
    fscanf(fread,"%d",&nodesayisi);
    fscanf(fread,"%d",&aracsayisi);
    for(i=0;i<aracsayisi+5;i++)
        dcap[i]=0;
    for(i=0;i<aracsayisi;i++)
    {
        fscanf(fread,"%d %d %lf", &dcap[i],&dcost[i],&alpha[i]);
        printf("%d %d %lf\n",dcap[i],dcost[i],alpha[i]);
    }
    for(i=0;i<nodesayisi;i++)
    {
        fscanf(fread," %d %lf %lf\n",&k,&x[i],&y[i]);
        printf("%d %lf %lf\n",i,x[i],y[i]);
    }
    for(i=0;i<nodesayisi;i++)
    {
        fscanf(fread," %d %d\n",&h,&capacity[i]);
        printf("%d %d\n",i,capacity[i]);
    }
    fclose(fread);
    nodesayisi--;
    for(i=0;i<=nodesayisi;i++)
        for(j=0;j<=nodesayisi;j++)
        {
            if(i==j)
                dist[i][j]=0;
            else
            {
                hold=sqrt(pow((x[i]-x[j]),2) + pow((y[i]-y[j]),2));
                dist[i][j]=hold;
            }
        }
    }
}

```

```

}

void readdistances(char * filename)
{
    double result,hold;
    int i,j,k,max;
    FILE *fread;
    FILE *fnum;
    FILE *fwei;
    fread = fopen(filename,"r");
    printf("dosyayy actim, %s\n",filename);
    fscanf(fread,"%d %d",&nodesayisi, &aracsayisi);
        fwei=fopen(filew,"r");
        fscanf(fwei,"%d",&max);
    for(i=0;i<aracsayisi;i++)
    {
        fscanf(fread,"%d",&dcap[i]);
        fscanf(fread,"%lf",&dcost[i]);
        printf("%d %lf",dcap[i],dcost[i]);
        if((nodesayisi>30)&&(i<max))
        {
            fscanf(fwei,"%lf",&alpha[i]);
        }
        else
        {
            alpha[i]=1;
        }
    }
    for(i=0;i<nodesayisi;i++)
    {
        fscanf(fread,"%d %lf %lf\n",&k,&x[i],&y[i]);
    }
    for(i=0;i<nodesayisi;i++)
    {
        fscanf(fread," %d %d\n",&k,&capacity[i]);
    }
    fclose(fread);
    for(i=0;i<nodesayisi;i++)
        for(j=0;j<nodesayisi;j++)
        {
            if(i==j)
                dist[i][j]=0;
            else
            {
                hold=sqrt(pow((x[i]-x[j]),2) + pow((y[i]-y[j]),2));
                if(hold*1000000-floor(hold*1000000)<0.5)
                    dist[i][j]=floor(hold*1000000)/1000000;
                else dist[i][j]=floor((hold+1)*1000000)/1000000;
            }
        }
    }
}

int fcheck (bool *r, int kac)
{
    int i,j,k;

```

```

int sayimiz,count=0;
if(kac==nodesayisi-1)
{
    sayimiz=1;
}
else
{
    sayimiz = rand() % (nodesayisi-kac);
    sayimiz++;
}
for(i=1;i<nodesayisi+1;i++)
{
    if(r[i]==false) count++;
    if(sayimiz == count)
    {
        r[i]=true;
        break;}
}
return i;
}

void initialize ()
{
    int i,j,k;
    bool * dolumu;
    int place;
    dolumu = new bool [nodesayisi+2];
    for(i=0;i<nodesayisi+1;i++)
        dolumu[i]=false;
    for(i=0;i<nodesayisi;i++)
    {
        place=fcheck(dolumu,i);
        ajan[i][1].no=place;
        ajan[i][1].cap=capacity[place];
        ajan[i][0].cap=capacity[place];
        doluluk[i]=1;
    }
    if(initializor==0)
        for(j=0;dcap[j]>=0;j++)
        {
            if(ajan[i][0].cap<=dcap[j])
            {
                vehiclecapacity[i]=dcap[j];
                break;
            }
        }
    if(initializor==2)
        for(j=0;dcap[j]>=0;j++)
        {
            if(ajan[i][0].cap<=dcap[j])
            {
                vehiclecapacity[i]=dcap[rand()%aracsayisi];
                break;
            }
        }
}

void findandbbwrite(int ajanno)
{
    int i, j, k;

```

```

double maximumsaver=0;
double temp=0;
int place = -1;
for(i=1;;i++)
{
    if(ajan[ajanno][i].no==0)
        break;
    else
    {
        temp=(dist[ajan[ajanno][i-1].no][ajan[ajanno][i].no] +
dist[ajan[ajanno][i].no][ajan[ajanno][i+1].no]) - dist[ajan[ajanno][i-1].no][ajan[ajanno][i+1].no];
        if(i==1)
        {
            maximumsaver=temp;
            place=i;
        }
        else
        {
            if(temp<=maximumsaver)
            {
                maximumsaver=temp;
                place=i;
            }
        }
    }
}
if(maximumsaver >= 0)
{
    if((blackb[ajan[ajanno][place].no].ownerid==ajanno) &&
(blackb[ajan[ajanno][place].no].savings==maximumsaver))
        dahaoncevermistim[ajanno]++;
    else
        dahaoncevermistim[ajanno] = 0;

    if(dahaoncevermistim[ajanno]>=4)
    {
        blackb[ajan[ajanno][place].no].ownerid=-1;
        blackb[ajan[ajanno][place].no].savings=-1;
    }
    else
    {
        blackb[ajan[ajanno][place].no].ownerid=ajanno;
        blackb[ajan[ajanno][place].no].savings=maximumsaver;
    }
}
}

void giveaprice(int ajanno)
{
    int i,j,k,stop = 0,duz,arac,gec,veh,hount,kucuk;
    double temp=-1, temp1=-1,cost, temp2=-1, min=-1,avg=0;
    int bestplace=-1;
    if(kandir==2)
    {
        avg=0;
        hount=0;
        for(i=0;i<nodesayisi;i++)
        {
            if((i!=ajanno) && ajan[ajanno][i].no!=0)

```

```

                {
                    avg = avg +
(double)ajan[i][0].cap/(double)vehiclecapacity[i];
                    hount++;
                }
            }
            avg=avg/hount;
        }
        for(i=0;i<=nodesayisi;i++)
        {
            for(j=0;j<=nodesayisi;j++)
            {
                if((i!= blackb[j].ownerid) && (blackb[j].savings>=0))
                {
                    duz=-1;
                    temp1=-1;
                    temp2=-1;
                    gec=1;
                    veh=-1;
                    cost=0;
                    if(ajan[i][1].no==0)
                    {
                        for(k=0;dcap[k]>=0;k++)
                        {
                            if(blackb[j].cap>=dcap[k])
                            {
                                cost=dcost[k];
                                kukuk=k-1;
                                break;
                            }
                        }
                    }
                    else
                    if((ajan[i][0].cap + blackb[j].cap)> vehiclecapacity[i])
                    {
                        for(k=0;dcap[k]>=0;k++)
                        {
                            if((dcap[k]==vehiclecapacity[i])
|| (veh!=-1))
                            {
                                veh=k;
                                kukuk=k-1;
                                if(dcap[k+1]<=0)
                                {
                                    keep[i][j].savings=-1;
                                    keep[i][j].place=-1;
                                    gec=0;
                                    break;
                                }
                                else
                                    if
                                (dcap[k+1]>=(ajan[i][0].cap + blackb[j].cap))
                                {
                                    cost=dcost[k+1]-
                                    kukuk=k;
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}
for(k=1;(k<=nodesayisi+1 && gec==1);k++)
{
    temp1=-dist[ajan[i][k-1].no][blackb[j].bas] -
dist[blackb[j].son][ajan[i][k].no] + dist[ajan[i][k-1].no][ajan[i][k].no] + blackb[j].savings;
    temp2=-dist[ajan[i][k-1].no][blackb[j].bas] -
dist[blackb[j].son][ajan[i][k].no] + dist[ajan[i][k-1].no][ajan[i][k].no] + blackb[j].savings;
    if(temp1>=temp2)
    {
        if((inicializor==1)&&(conv<=10))
            temp=temp1*alpha[k+1];
        else
            temp=temp1*alpha[k+1]-cost;
        duz=1;
    }
    else
    {
        if((inicializor==1)&&(conv<=10))
            temp=temp2*alpha[k+1];
        else
            temp=temp2*alpha[k+1]-cost;

        duz=0;
    }
    if(temp> min)
    {
        min = temp;
        bestplace = k;
    }
    if(ajan[i][k].no==0) break;
}
if(min>=0)
{
    if(kandir==2)
    {
        min=min *
pow(((double)(ajan[ajanno][0].cap)/((double)vehiclecapacity[ajanno])),((double)run)/10);
    }

    keep[i][j].savings = min;
    keep[i][j].place = bestplace;
    keep[i][j].yon=duz;
}
else
{
    keep[i][j].savings=-1;
    keep[i][j].place=-1;
}
temp=-1;
bestplace=-1;
min=-1;
}
else
{
    keep[i][j].savings=-8;
    keep[i][j].place=-8;
}

```



```

    }
}
}
void sendToCentralAgent(void)
{
    int i,j,k;

    if((selector>=1) && (selector<2))
        selectranking(selector);
    else
        selectbest();
    for(i=0;i<=nodesayisi+1;i++)
    {
        blackb[i].savings=-10;
        blackb[i].sayi=0;
        blackb[i].cap=0;
    }
}
void selectbest(void)
{
    int i,j,k;
    int eniyiajan, eniyisehir,verenajan;
    bool assigned[300];
    for(i=0;i<nodesayisi+2;i++)
        assigned[i]=false;
    double max;
    for(j=1;j<=nodesayisi;j++)
    {
        max=-0.5;
        for(i=0;i<nodesayisi;i++)
        {
            if((keep[i][j].savings>= max) && assigned[i]==false)
            {
                max = keep[i][j].savings;
                eniyiajan=i;
                assigned[i]=true;
            }
        }
        if(max>=0 )
        {
            update(j,j,enyiajan,keep[enyiajan][j].place);
        }
    }
}

void update(int sira, int verenajan, int alanajan, int konacakyer)
{
    int i,j,k,tut,go1,go2;
    int flag=0,clack=0,gec=0,tec=0;
    int hold,cold,thold,tcold,say=0;
    double cost;
    if(((tradeyaptim[alanajan]==0) && (tradeyaptim[verenajan]==0))&&
    (ajan[verenajan][1].no!=0)&&((blackb[verenajan].cap==ajan[verenajan][0].cap)&&(ajan[alanajan][1
].no!=0))&&(blackb[verenajan].bas!=0))
    {
        cost=0;
        for(j=0;j<=nodesayisi+1;j++)
        {

```

```

        if((ajan[verenajan][j+1].no==blackb[verenajan].bas) && (clack !=1))
        {
            flag=1;
            tut=ajan[verenajan][0].cap;
            ajan[verenajan][0].cap=ajan[verenajan][0].cap
blackb[verenajan].cap;
            for(k=0;;k++)
            {
                ajan[299][k].no=ajan[verenajan][j+k+1].no;
                ajan[299][k].cap=ajan[verenajan][j+k+1].cap;
                say++;
                if(ajan[299][k].no==blackb[verenajan].son)
                    break;
            }
        }
        else if(flag==1)
        {
            for(k=0;;k++)
            {
                ajan[verenajan][j+k].no=ajan[verenajan][j+say+k].no;
                ajan[verenajan][j+k].cap=ajan[verenajan][j+say+k].cap;
                if (k>=say)
                    break;
            }
        }
        if(ajan[verenajan][j+1].no==0)
            break;
    }
    ajan[alanajan][0].cap=ajan[alanajan][0].cap + blackb[verenajan].cap;
    if(keep[alanajan][konacaker].yon==1)
    {
        for(k=0;k<say;k++)
        {
            gec=ajan[alanajan][konacaker+k].no;
            tec=ajan[alanajan][konacaker+k].cap;
            ajan[alanajan][konacaker+k].no=ajan[299][k].no;
            ajan[alanajan][konacaker+k].cap=ajan[299][k].cap;
            ajan[299][k].no=gec;
            ajan[299][k].cap=tec;
        }
    }
    else
    {
        for(k=0;k<say;k++)
        {
            gec=ajan[alanajan][konacaker+k].no;
            tec=ajan[alanajan][konacaker+k].cap;
            ajan[alanajan][konacaker+k].no=ajan[299][say-k-1].no;
            ajan[alanajan][konacaker+k].cap=ajan[299][say-k-1].cap;
            ajan[299][say-k-1].no=gec;
            ajan[299][say-k-1].cap=tec;
        }
    }
    for(k=doluluk[alanajan]+say;k>=konacaker+say;k--)
    {
        ajan[alanajan][k].no=ajan[alanajan][k-say].no;
        ajan[alanajan][k].cap=ajan[alanajan][k-say].cap;
    }

```

```

}
for(k=konacaker+say;k<konacaker+2*say;k++)
{
    ajan[alanajan][k].no=ajan[299][-konacaker-say+k].no;
    ajan[alanajan][k].cap=ajan[299][-konacaker-say+k].cap;
}
tradeyaptim[alanajan]=1;
tradeyaptim[verenajan]=1;
doluluk[alanajan]=doluluk[alanajan]+say;
doluluk[verenajan]=doluluk[verenajan]-say;
if(!((inializor==1)&& (conv<=10)))
{
    vehiclecapacity[verenajan]=0;
    vehiclecapacity[alanajan]=0;

    for(i=1;i<300;i++)
    {
        vehiclecapacity[verenajan] = vehiclecapacity[verenajan] +
ajan[verenajan][i].cap;
        if(ajan[verenajan][i+1].cap==0)
            break;
    }

    for(i=1;i<300;i++)
    {
        vehiclecapacity[alanajan] = vehiclecapacity[alanajan] +
ajan[alanajan][i].cap;
        if(ajan[alanajan][i+1].cap==0)
            break;
    }
    if(ajan[verenajan][1].cap==0)
        go1=0;
    else go1=1;
    go2=1;
    for(k=0;dcap[k]>0;k++)
    {
        if(k==0)
            if((vehiclecapacity[verenajan]>0
vehiclecapacity[verenajan]<=dcap[k]) && (go1==1))
                {
                    vehiclecapacity[verenajan]=dcap[k];
                    go1=0;
                }
            if((vehiclecapacity[alanajan]>0
vehiclecapacity[alanajan]<=dcap[k]) && (go2==1))
                {
                    vehiclecapacity[alanajan]=dcap[k];
                    go2=0;
                }
            else
                if((vehiclecapacity[verenajan]>dcap[k-1]
vehiclecapacity[verenajan]<=dcap[k]) && (go1==1))
                    {
                        vehiclecapacity[verenajan]=dcap[k];
                        go1=0;
                    }

                if((vehiclecapacity[alanajan]>dcap[k-1]
vehiclecapacity[alanajan]<=dcap[k]) && (go2==1))

```

```

        {
            vehiclecapacity[alanajan]=dcap[k];
            go2=0;
        }
        if((go1==0) && (go2==0))
            break;
    }
}

void improve(int ajanno)
{
    int i,j,k,temp,cemp;
    double old,pro, pro1,pro2;
    for(i=0;i<=nodesayisi-1;i++)
    {
        for(j=i+2;j<nodesayisi+1;j++)
        {
            if(ajan[ajanno][i].no==ajan[ajanno][j+1].no) break;
            if(ajan[ajanno][j].no==0) break;
            old = dist[ajan[ajanno][i].no][ajan[ajanno][i+1].no] +
dist[ajan[ajanno][j].no][ajan[ajanno][j+1].no];
            pro1 = dist[ajan[ajanno][i].no][ajan[ajanno][j+1].no] +
dist[ajan[ajanno][j].no][ajan[ajanno][i+1].no];
            pro2 = dist[ajan[ajanno][i].no][ajan[ajanno][j].no] +
dist[ajan[ajanno][i+1].no][ajan[ajanno][j+1].no];
            if(( pro1 < pro2 ) && (old >= pro1))
            {
                temp=ajan[ajanno][i+1].no;
                ajan[ajanno][i+1].no=ajan[ajanno][j+1].no;
                ajan[ajanno][j+1].no=temp;
                cemp=ajan[ajanno][i+1].cap;
                ajan[ajanno][i+1].cap=ajan[ajanno][j+1].cap;
                ajan[ajanno][j+1].cap=cemp;
            }
            else if(old>=pro2)
            {
                temp=ajan[ajanno][j].no;
                ajan[ajanno][j].no=ajan[ajanno][i+1].no;
                ajan[ajanno][i+1].no=temp;
                cemp=ajan[ajanno][i+1].cap;
                ajan[ajanno][i+1].cap=ajan[ajanno][j+1].cap;
                ajan[ajanno][j+1].cap=cemp;
            }
        }
    }
}

void hepsiniduyur(int ajanno)
{
    int i, j, k;
    double maximumsaver=0;
    double temp=0;
    int place = -1;
    for(i=1;;i++)
    {
        if(ajan[ajanno][i].no==0)

```

```

        break;
    else
    {
        temp=(dist[ajan[ajanno][i-1].no][ajan[ajanno][i].no] +
dist[ajan[ajanno][i].no][ajan[ajanno][i+1].no] - dist[ajan[ajanno][i-1].no][ajan[ajanno][i+1].no];
        if(temp >= 0)
        {
            if((blackb[ajan[ajanno][i].no].ownerid==ajanno) &&
(blackb[ajan[ajanno][i].no].savings==maximumsaver))
                dahaoncevermistim[ajanno]++;
            else
                dahaoncevermistim[ajanno] = 0;

            if(dahaoncevermistim[ajanno]>=4)
            {
                blackb[ajan[ajanno][i].no].ownerid=-1;
                blackb[ajan[ajanno][i].no].savings=-1;
            }
            else
            {
                blackb[ajan[ajanno][i].no].ownerid=ajanno;
                blackb[ajan[ajanno][i].no].savings=temp;
            }
        }
    }
}
}
}

```

```

void selectranking(double etha)
{
    int i,j,k,count,current,temp;
    int feasibles[300];
    double probability[300];
    bool assigned[300];
    for(i=0;i<nodesayisi+2;i++)
        assigned[i]=false;
    double attim;
    for(j=0;j<nodesayisi+1;j++)
    {
        count=0;
        for(i=0;i<nodesayisi;i++)
            if(keep[i][j].savings>=0)
            {
                feasibles[count]=i;
                count++;
            }
        for(i=0;i<count;i++)
        {
            current=feasibles[i];
            for(k=0;k<i;k++)
            {
                if(keep[current][j].savings > keep[feasibles[k]][j].savings)
                {
                    temp=feasibles[k];
                    feasibles[k]=current;
                    current=temp;
                }
            }
        }
    }
}

```

```

        }
        feasibles[k]=current;
    }
    for(i=0;i<count;i++)
    {
        probability[i]=((double)1/(double)count)*(etha-(2*(etha-
1)*((double)(i))/(double)(count-1)))));
    }
    attim = ( (double)rand() / ((double)(RAND_MAX)+(double)(1)) );
    for(i=0;i<count;i++)
    {
        if(attim<=probability[i] && assigned[feasibles[i]]!=true)
        {
            update(j,j,feasibles[i],keep[feasibles[i]][j].place);
            assigned[feasibles[i]]=true;
            break;
        }
        else attim=attim-probability[i];
    }
}
}

void find2write(int ajanno)
{
    int i, j, k, t ,zap,oran, ilk, son,kap,sayi,hount,eskibas,eskison,kucuk;
    double maximalsaver=0,kar;
    double temp=0,avg;
    int place = -1;
    eskibas=blackb[ajanno].bas;
    eskison=blackb[ajanno].son;
    for(i=1;;i++)
    {
        kar=0;
        zap=0;
        if(ajan[ajanno][i].no==0)
        {
            blackb[ajanno].bas=-1;
            blackb[ajanno].son=-1;
            break;
        }
        else
        {
            for(j=i;ajan[ajanno][j].no!=0;j++)
            {
                temp=(dist[ajan[ajanno][i-1].no][ajan[ajanno][i].no] +
dist[ajan[ajanno][j].no][ajan[ajanno][j+1].no]) - dist[ajan[ajanno][i-1].no][ajan[ajanno][j+1].no];
                zap=zap+ajan[ajanno][j].cap;
                if(method==2)
                if((dahaoncevermistim[ajanno]>=4) &&
(blackb[ajanno].bas==ilk) && (blackb[ajanno].son==son))
                {
                    i++;
                    temp=(dist[ajan[ajanno][i-1].no][ajan[ajanno][i].no] +
dist[ajan[ajanno][i].no][ajan[ajanno][i+1].no]) - dist[ajan[ajanno][i-1].no][ajan[ajanno][i+1].no];
                }
                for(k=0;dcap[k]>=0;k++)
                {
                    if(vehiclecapacity[ajanno]==dcap[k])

```

```

        {
            kukuk=k-1;
            break;
        }
    }
    if(ajan[ajanno][0].cap-zap==0)
    {
        kar=dcost[k];
    }

else if(vehiclecapacity[ajanno]-zap <= dcap[k-1])
    {
        for(t=0;dcap[t]<=vehiclecapacity[ajanno]-zap;t++);
        kar=-dcost[k]+dcost[t];
    }
if(i==1)
    {
        if((initializor==1) && (conv<=10))
            maximumsaver=alpha[kucuk+1]*temp;
        else
            maximumsaver=alpha[kucuk+1]*temp+kar;
        place=i;
        oran=maximumsaver/(j-i+1);
        sayi=j-i+1;
        ilk=ajan[ajanno][i].no;
        son=ajan[ajanno][j].no;
        kap=zap;

    }
else
    {
        if((oran <= temp/(j-i+1)))
        {

            if((initializor==1) && (conv<=10))
                maximumsaver=alpha[kucuk+1]*temp;
            else
                maximumsaver=alpha[kucuk+1]*temp+kar;
            place=i;
            oran=temp/(j-i);
            ilk=ajan[ajanno][i].no;
            son=ajan[ajanno][j].no;
            kap=zap;
            sayi=j-i+1;

        }
    }
}
}
if(maximumsaver >= 0)
{
    if((eskibas==ilk) && (eskison==son))
        dahaoncevermistim[ajanno]++;
    else
        dahaoncevermistim[ajanno] = 0;

    if(kandir>0)
    {

```

```

        avg=0;
        hount=0;
        for(i=0;i<nodesayisi;i++)
        {
            if((i!=ajanno) && ajan[ajanno][1].no!=0)
            {
                avg = avg +
(double)ajan[ajanno][0].cap/(double)vehiclecapacity[ajanno];
                hount++;
            }
        }
        avg=avg/hount;

        if((dahaoncevermistim[ajanno]>=1) )

            maximumsaver=maximumsaver*log(1+((double)dahaoncevermistim[ajanno]+1)*((double)v
ehiclecapacity[ajanno])/((double)ajan[ajanno][0].cap));
        }

        blackb[ajanno].ownerid=ajanno;
        blackb[ajanno].savings=maximumsaver;
        blackb[ajanno].bas=ilk;
        blackb[ajanno].son=son;
        blackb[ajanno].cap=kap;
        blackb[ajanno].sayi=sayi;
    }
}
void findandwrite(int ajanno)
{
    int i, j, k;
    double maximumsaver=0,avg=0;
    double temp=0;
    int place = -1;

    for(i=0;i<nodesayisi;i++)
    {
        if(i!=ajanno)
            avg = avg + ajan[i][0].cap;
    }
    avg=avg/nodesayisi;
    for(i=1;;i++)
    {
        if(ajan[ajanno][i].no==0)
            break;
        else
        {
            temp=(dist[ajan[ajanno][i-1].no][ajan[ajanno][i].no]
dist[ajan[ajanno][i].no][ajan[ajanno][i+1].no]) - dist[ajan[ajanno][i-1].no][ajan[ajanno][i+1].no];
            if(i==1)
            {
                maximumsaver=temp;
                place=i;
            }
            else
            {
                if(temp<=maximumsaver)
                {
                    maximumsaver=temp;
                    place=i;
                }
            }
        }
    }
}

```



```

    }
    }
}
if(maximumsaver >= 0)
{
    if((blackb[ajan[ajanno][place].no].ownerid==ajanno) &&
(blackb[ajan[ajanno][place].no].savings==maximumsaver))
        dahaoncevermistim[ajanno]++;
    else
        dahaoncevermistim[ajanno] = 0;

        blackb[ajan[ajanno][place].no].ownerid=ajanno;
    if(ajan[ajanno][0].cap<=avg)
    {

        blackb[ajan[ajanno][place].no].savings=maximumsaver*log(1+((double)dahaoncevermistim
[ajanno]+1)*((double)vehiclecapacity[ajanno])/((double)ajan[ajanno][0].cap));
    }
    else
        blackb[ajan[ajanno][place].no].savings=maximumsaver;
}
}
void givesprice(int ajanno)
{
    int i,j,k,stop = 0;
    double temp=-1, min=-1,avg=0;
    int bestplace=-1;
    for(i=0;i<nodesayisi;i++)
    {
        if(i!=ajanno)
            avg = avg + ajan[i][0].cap;
    }
    avg=avg/nodesayisi;
    for(i=0;i<=nodesayisi+1;i++)
    {

        for(j=1;j<=nodesayisi;j++)
        {
            if((i!= blackb[j].ownerid) && (i!=-1))
            {
                if((ajan[i][0].cap + capacity[j])> vehiclecapacity[i])
                {
                    keep[i][j].savings=-1;
                    keep[i][j].place=-1;
                }
                else
                for(k=1;k<=nodesayisi+1 ;k++)
                {
                    temp=-dist[ajan[i][k-1].no][j] - dist[j][ajan[i][k].no] +
dist[ajan[i][k-1].no][ajan[i][k].no] + blackb[j].savings;
                    if(temp> min)
                    {
                        min = temp;
                        bestplace = k;
                    }
                    if(ajan[i][k].no==0) break;
                }
            }
        }
    }
}

```

```

    }
    if(min>=0 && ajan[ajanno][0].cap>0)
    {
        if(((double)ajan[ajanno][0].cap <= avg) &&
(ajan[ajanno][0].cap>0))
            {
                keep[i][j].savings= min *
pow(((double)(ajan[ajanno][0].cap)/((double)vehiclecapacity[ajanno])),((double)run)/10);
            }
            else
                keep[i][j].savings= min;
                keep[i][j].place=bestplace;
            }
        else
        {
            keep[i][j].savings=-1;
            keep[i][j].place=-1;
        }
        temp=-1;
        bestplace=-1;
        min=-1;
    }
    else
    {
        keep[i][j].savings=-8;
        keep[i][j].place=-8;
    }
}
}
}
}
}

```