

VEHICLE ROUTING PROBLEM WITH PICKUP AND DELIVERY AND WITH
TIME WINDOWS

by

Senem Kalyoncu

B.S., Industrial Engineering, İstanbul University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Bogaziçi University

2008

ACKNOWLEDGEMENTS

I would like to thank to my thesis supervisor Prof. Gülay Barbarosoğlu who guided me in every step of my studies with her valuable knowledge and experiences and also with her confidence which enabled me to complete my studies in the best way.

I am grateful to Prof. Tülin Aktin and Assoc. Prof. Necati Aras for their interest and willingness to serve as my committee member.

I am also grateful to TÜBİTAK for their financial support during my master education.

I am greatly indebted to my family for all their understanding, moral support and love.

I would like to express my sincere gratitude to Engin Sansarcı for his help by giving me new ideas and sharing his knowledge with me at the hard periods of my studies.

I would like to thank to my best friend Zeynep Sinem Polat and all my friends for believing and motivating me by enthusiasm.

Furthermore I would like to specify that it is a great pleasure and honor for me to make my thesis in Boğaziçi University and to be a part of this family.

ABSTRACT

VEHICLE ROUTING PROBLEM WITH PICKUP AND DELIVERY AND WITH TIME WINDOWS

This work proposes a metaheuristic method for solving Vehicle Routing Problem with Pickup and Delivery with Time Windows (VRPPDTW). The mathematical model which is developed to solve this problem tries to fulfill a set of transportation requests within time intervals of related nodes of their request by a heterogeneous vehicle fleet based at multiple depots so as to minimize the total travel cost.

A mixed integer model is developed for this problem, and a Simulated Annealing algorithm (SA) is designed as the solution procedure which employs two types of moves, namely Request Exchange and Merge.

The algorithm is experimented on a set of randomly generated instances. The instances are also solved by CPLEX solver within GAMS suite v22.0. The results are compared. It is observed that CPLEX could find solutions for only small-size instances in reasonable computing time. The algorithm developed in this thesis is capable of solving large size problems in a reasonable amount of time.

Then the SA algorithm is applied to Boğaziçi University (BU) Personnel Transportation System to determine the optimal schedule for transporting groups of personnel from specific pickup locations in different parts of the Metropolitan city of İstanbul to the delivery location, i.e. the main university campus in Bebek.

ÖZET

ZAMAN KISITLI DAĞITIM VE TOPLAMA GÜZERGAHI BULMA PROBLEMİ

Bu çalışmada Zaman Kısıtlı Dağıtım ve Toplama Güzergahı Bulma problemi için bir optimizasyon metodu geliştirilmiştir. Bu problem tipini çözmek için geliştirilen matematiksel modelde, farklı kapasitelere sahip araçlar, farklı depolardan yola çıkarak, toplama noktalarından aldıkları malı, ilgili dağıtım noktasına teslim ederler. Herbir noktaya belirli zaman aralığı içerisinde hizmet vermeleri gerekmektedir.

Problemın çözümü için bir tamsayılı model geliştirilmiştir. Çözümü için tavlama benzetimli bir algoritma geliştirilmiştir. Bu algoritmada çözüm alanı içerisinde komşuluk aramak için iki tip strateji kullanılmıştır. Bunlar, iki farklı talebin toplama ve dağıtım noktalarının birbirleri ile yer değiştirmesi ve farklı araçlar tarafından hizmet verilen taleplerin birleştirilmesidir.

Geliştirilen algoritma rastgele üretilmiş problemler üzerinde denenmiştir. Aynı problemler GAMS v22.0 de CPLEX çözücüsü tarafından da çözülmüş ve sonuçlar karşılaştırılmıştır. Küçük ölçekli problemler için CPLEX in en iyi sonuçları makul bir zaman içerisinde verdiği gözlemlenmiştir. Fakat büyük ölçekli problemler için aynı başarıyı elde edememiştir. Bu tezde geliştirilen algoritma büyük ölçekli problemleri makul bir zaman içerisinde çözebilir niteliktedir.

Son olarak geliştirilen algoritma ile Boğaziçi Üniversitesi'nin servis problemi için en iyi çözüm bulunmaya çalışılmıştır. Metropolitan bir şehir olan İstanbul'un farklı semtlerinde ikamet eden personellerin bulundukları noktadan alınıp Bebek'te bulunan ana kampüse getirilmesine ilişkin problem çözülmüştür.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	viii
LIST OF TABLES	ix
LIST OF SYMBOLS/ABBREVIATIONS	xi
1. INTRODUCTION	1
2. LITERATURE REVIEW AND BACKGROUND	3
2.1. Vehicle Routing Problem (VRP)	3
2.2. VRP with Time Windows	4
2.3. VRP with Pickup and Delivery	5
2.4. VRP with Pickup and Delivery with Time Windows	6
3. PROBLEM DESCRIPTION	10
3.1. Model Formulation	10
3.1.1. Inputs of the Model	11
3.1.2. Main Assumptions	11
3.1.3. Outputs of the Model	12
3.1.4. Alternative Objective Functions	12
3.1.5. Parameters	13
3.1.6. Decision Variables	13
3.2. Mathematical Formulation of the Model	14
3.3. Dimensionality of the Model	17
4. THE SOLUTION APPROACH	18
4.1. Combinatorial Problems	18
4.2. Classical Heuristics for Solving Vehicle Routing Problems	19
4.2.1. Constructive Heuristics	20
4.2.2. Two-phase heuristics	21
4.2.3. Improvement Heuristics	22
4.3. Motivation for MetaHeuristic Approaches	22

4.4. Overview of General Simulated Annealing Metaheuristic Structure . . .	24
4.5. Description of the SA Algorithm	26
5. EXPERIMENTAL DESIGN	35
6. REAL-LIFE APPLICATION	48
7. CONCLUSIONS	51
7.1. Summary	51
7.2. Recommendations for Future Research	51
APPENDIX A: RESULTS OF EXPERIMENTAL DESIGN	53
APPENDIX B: DATA SET OF BU PROBLEM	60
APPENDIX C: THE CODE OF SA ALGORITHM	68
REFERENCES	118

LIST OF FIGURES

Figure 2.1.	The basic problems of the VRP class and their interconnections	4
Figure 4.1.	A conceptual representation of global and local optimum	23
Figure 4.2.	Move type of request exchange	30
Figure 4.3.	Move type of merge	30
Figure 5.1.	Iteration 1 of the solution approach	40
Figure 5.2.	Iteration 2 of the solution approach	40
Figure 5.3.	Iteration 3 of the solution approach	40
Figure 5.4.	Iteration 4 of the solution approach	41
Figure 5.5.	Iteration 5 of the solution approach	41
Figure 5.6.	Iteration 6 of the solution approach	41

LIST OF TABLES

Table 4.1.	The pseudocode of the local neighborhood search.....	24
Table 4.2.	General structure of the SA algorithms	26
Table 4.3.	The pseudocode of the SA algorithm	33
Table 5.1.	Ranges of the parameter values for different problem sizes	38
Table 5.2.	Data set of the 4 PD problem	40
Table 5.3.	Vehicle data set of the 4 PD problem	40
Table 5.4.	Results of the SA heuristic with different probability functions for 4 PD	43
Table 5.5.	Results of the SA heuristic with different probability functions for 5 PD	44
Table 5.6.	Results of the SA heuristic with different probability functions for 6 PD	44
Table 5.7.	Results of the SA heuristic with different probability functions for 10 PD	45
Table 5.8.	Results of the SA heuristic with different probability functions for 20 PD	45
Table 5.9.	Results of the SA heuristic with different probability functions for 50 PD	45
Table 5.10.	Results of the SA heuristic with different probability functions for 75 PD	46

Table 5.11.	Results of the SA heuristic with different probability functions for 100 PD	46
Table 5.12.	The summary of the results	46
Table 6.1.	The solution of BU personel transportation system	51
Table A.1.	Routes found by CPLEX and the SA heuristic for 4 PD	54
Table A.2.	Routes found by CPLEX and the SA heuristic for 5 PD	55
Table A.3.	Routes found by CPLEX and the SA heuristic for 6 PD	56
Table A.4.	Routes found by the SA heuristic for 10 PD	57
Table A.5.	Routes found by the SA heuristic for 20 PD	58
Table A.6.	Routes found by the SA heuristic for 50 PD	59
Table A.7.	Routes found by the SA heuristic for 75 PD	60
Table A.8.	Routes found by the SA heuristic for 100 PD	60
Table B.1.	Data set of BU personnel transportation system	61
Table B.2.	The capacities of buses in BU personnel transportation system	68

LIST OF SYMBOLS/ABBREVIATIONS

α	Coefficient for temperature reduction function in SA
β	Coefficient for temperature temperature iteration function
δ	Difference between the objective function values of neighboring solutions
BHS	Best solution found by the heuristic algorithm
BU	Boğaziçi University
CVRP	Capacitated Vehicle Routing Problem
D	Delivery node
DARP	Dial-a-Ride Problem
DCVRP	Distance-Constrained Vehicle Routing Problem
ET	Elapsed Time
f	Objective function value
$f_{current}$	The objective function value of the current solution
GPDP	General Pickup and Delivery Problems
$iter_{algorithm}$	Current iteration number in the SA algorithm
$iter_{T_r}$	Iteration number in the current temperature in SA
k	A preassigned iteration number for finding initial temperature
LTW	Latest time window
$\max_{iter_algorithm}$	A preassigned iteration number used for termination
$\max_{iter_temp}$	Maximum iteration number of the current temperature
$NITER$	Number of iteration
NP	Non-deterministic Polynomial Time
NPD	Number of pickup and delivery nodes
NPI	Number of problem instances
NV	Number of vehicles
P	Pickup node
P_a	Acceptance probability function
PDP	Pickup and Delivery Problem

PDTSP	Pickup and Delivery Traveling Salesman Problem
PDVRP	Pickup and Delivery Vehicle Routing Problem
R	Request
SA	Simulated Annealing
$SOLN$	Solution
SP	Solution pool of initial solutions
T_{begin}	Initial temperature
T_r	Current temperature in the SA algorithm
$T_{threshold}$	Threshold temperature used for termination of the SA
TS	Tabu Search
TSP	Traveling Salesman Problem
VC	Vehicle capacity
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPPDTW	Vehicle Routing Problem with Pickup and Delivery with Time Windows
VRPTW	Vehicle Routing Problem with Time Windows

1. INTRODUCTION

Many problems in distribution management can be formulated as Vehicle Routing Problem with Pickup and Delivery with Time Windows since they naturally exhibit pickup and delivery constraints and time constraints in their modeling. These include dial-a-ride problem, airline scheduling, bus routing, street cleaning, logistics and maintenance support. In the last decades, an increasing utilization of optimization packages based on Operations Research and Mathematical Programming techniques for supply of goods and services in distribution systems have been observed. Substantial savings in the global transportation costs have been gathered. Since the proposed models consider all characteristics of the distribution problems arising in real-world applications and corresponding algorithms and computer implementations find good solutions for real-world instances within acceptable computing times.

VRPPDTW is the generalization and integration of both Vehicle Routing Problem with Pickup and Delivery and Vehicle Routing Problem with Time Windows. In this type of problems, a set of routes have to be constructed to satisfy transportation requests under given constraints. A fleet of vehicles is available to operate the routes. Each vehicle has a given capacity, a start and a destination depot. Each transportation request represents the size of the load to be transported from the picked up locations to corresponding delivered locations within given time windows. Loading and unloading times are incurred at each location. Each pickup and delivery location is visited exactly once and each pickup location is visited before its associated delivery location. Each request has to be fulfilled by the same vehicle. A delivery process should not be followed immediately by its picked up process.

VRPPDTW problems are NP-hard problems and optimal methods are not appropriate to solve the large versions of this problem in a reasonable amount time. heuristic algorithms are often developed to solve these problems.

This thesis proposes an SA algorithm that fulfills problem specifications men-

tioned above. The overall structure of the algorithm is motivated by minimizing the total travel cost which also indirectly decreases the number of vehicles. In the algorithm "Request Exchange" and "Merge" operations are designed as two main methods for constructing the consecutive neighbors and four different acceptance probability functions are applied.

The algorithm is also then applied to Boğaziçi University Personnel Transportation System to find the optimal schedule for transporting groups of personnel from specific pickup locations to the delivery location, i.e. the campus located in Bebek. Since the obtained solution is not appropriate for real-life application, some modifications are done.

The thesis is organized as follows: Literature review related to the problem is in Chapter 2. In Chapter 3, the problem is defined and the mathematical model is formulated. Chapter 4 describes the SA algorithm designed to solve the problem under consideration. Chapter 5 contains the experimental study. In Chapter 6, the real-life application applied is described and the solution is illustrated. Chapter 7 emphasizes the main characters of the study and gives recommendations for future research related to this study.

2. LITERATURE REVIEW AND BACKGROUND

2.1. Vehicle Routing Problem (VRP)

The problems including the transportation of goods between depots and customers are generally known as Vehicle Routing Problems (VRPs). This type of problem does not only consider transportation of goods but different real-world applications in transportation systems, too. Typical applications of this type problem could be listed as solid waste collection, street cleaning, school bus routing, dial-a-ride systems, transportation of handicapped persons, routing of salespeople and of maintenance units.

The solution of VRP determines a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all the requirements of the customers are fulfilled, all the operational constraints are satisfied, and the global transportation cost is minimized.

In the last decades, an increasing utilization of optimization packages based on Operations Research and Mathematical Programming techniques for supply of goods and services in distribution systems have been observed. Substantial savings in the global transportation costs have been gathered in the large number of real-world applications that have used these optimization packages.

The main factor of success is the development of modeling and algorithmic tools implemented in recent years. Since the proposed models consider all characteristics of the distribution problems arising in real-world applications and corresponding algorithms and computer implementations find good solutions for real-world instances within acceptable computing times.

Different problem types can be identified within the context of VRP. These are Capacitated VRP (CVRP), Distance-Constrained VRP (DCVRP), the VRP with Time Windows (VRPTW), the VRP with Backhauls (VRPB) and the VRP with Pickup

and Delivery (VRPPD). Figure (2.1) summarizes these problems and illustrates their connections.

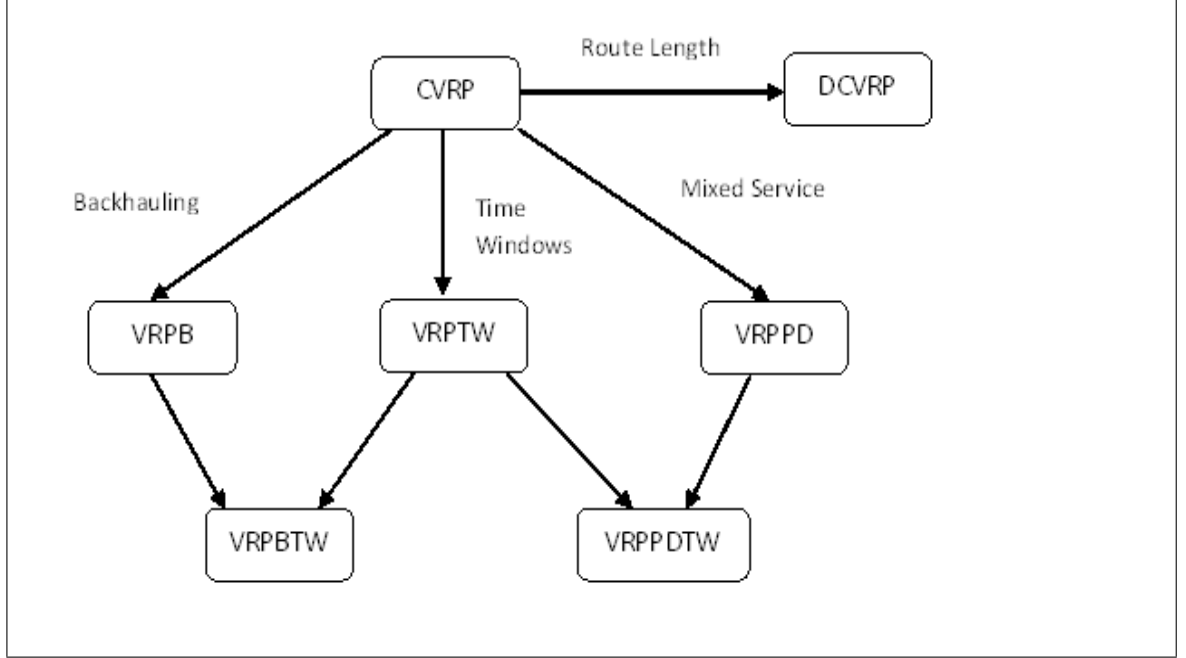


Figure 2.1. The basic problems of the VRP class and their interconnections

As it is seen , VRPPDTW (Vehicle Routing Problem with Pickup and Delivery with Time Windows) is a generalization of VRPTW or VRPPD . A comprehensive summary is provided in Paolo and Daniele (2001).

2.2. VRP with Time Windows

The VRPTW is the extension of the CVRP where the service at each customer must start within an associated time window and the vehicle must remain at the customer location during service. Soft time windows can be violated at a cost, while hard time windows do not allow for a vehicle to arrive at a customer after the latest time to begin service. In the latter case, generally if it arrives before the customer is ready to begin service, it waits.

Early work on the VRPTW often utilized simple heuristics or metaheuristics in local search, and a comprehensive summary can be found in Gendreau et al. (1997). In recent years, more complicated metaheuristics have been utilized to obtain higher

quality results. These include simulated annealing (Chiang and Russell 1996), tabu search (Rochat and Taillard 1995, Taillard et al. 1997, Chiang and Russell 1997, De Backer et al. 2000, Cordeau et al. 2001), genetic/evolutionary algorithms (Potvin and Beigo 1996, Homberger and Gehring 1999, Gehring and Homberger 2001), utilization of simulated annealing, tabu search and genetic algorithm (Tan et al. 2001), large neighborhood search (Shaw 1998), hybrid of simulated annealing and large neighborhood search (Russell and Pascal 2004), ant colony search (Gambardella et al. 1999), and variable neighborhood search (Rousseau et al. 2002).

2.3. VRP with Pickup and Delivery

General Pickup and Delivery Problems (GPDP) have two type classes. One is Vehicle Routing Problem with Backhauls (VRPB) in which all customer deliveries originate at a unique depot and all collections are destined to the depot. VRPPD is the other class of GPDP in which all deliveries are performed between customers.

In the VRPPD problems, a heterogeneous vehicle fleet based at multiple terminals must satisfy a set of transportation requests. Each request is defined by a pickup node, a corresponding delivery node and a demand to be transported between these locations. The requested transport could involve goods or persons. The constraints of problems in this class are visiting each pickup and delivery stop exactly once, not exceeding the capacity of vehicles, and coupling the pickup and corresponding delivery stops on the same vehicle routes and visit precedence among each pickup stop and its associated drop-off stop. There are also depot constraints that ensure vehicles return to the appropriate terminals and resource restrictions on the number of drivers and vehicle types.

VRPPD problems are broadly divided into 2 classes: The first class refers to situations where pickup and delivery locations are unpaired. Each unit picked up can be used fulfill the demand of any deliver customer. The second class refers to situations where pickup and delivery locations are paired. First class includes Pickup and Delivery Vehicle Routing Problem (PDVRP) and Pickup and Delivery Traveling Salesman

Problem (PDTSP), in multi and in the single vehicle case, respectively. The second class includes the classical pickup and delivery problems (PDPs) in which goods are considered and the Dial-A-Ride Problems (DARPs) in which passenger transportation and user convenience are taken into consideration.

The solution methods for VRPPD which include exact methods, heuristics and metaheuristics could be found in the survey studies of Parragh et al. (2006) and Berbeglia et al. (2007).

2.4. VRP with Pickup and Delivery with Time Windows

It is a generalization of VRPPD and VRPTW. In VRPPDTW, a heterogeneous vehicle fleet based at multiple depots must satisfy a set of transportation requests within time intervals of related nodes of the request.

The work in the VRPPDTW can be classified in different ways. One possible is to classify mainly into 2 classes: studies on pickup and delivery problem with time windows and studies on DARP with time windows. The studies related to first class are as follows:

Van der Bruggen (1993) considered single vehicle PDPTW with the objective of minimizing of route duration. They proposed a local search method based on a variable-depth search. The method consists of two-phases. In the first phase, a feasible route is constructed. In the second phase, it is iteratively improved.

Ioachim et al. (1995) proposed an approximation algorithm for the multiple-vehicle VRPPDTW. The algorithm first creates a set of clusters by column generation and then a multiple Traveling Salesman Problem (TSP) is solved to assign clusters into the final routes.

Nanry and Barnes (1998) developed a reactive tabu search approach for multiple-vehicle VRPPDTW. They used 3 distinct move neighborhoods that strictly enforce

precedence and coupling constraints so they penalized time window and load violations. They constructed a new set of benchmark problems for the pickup and delivery problem with time windows based on Solomon's benchmark vehicle routing problem with time windows data sets.

Landrieu et al. (2001) developed a probabilistic tabu search to solve the single-vehicle VRPPDTW and they applied their algorithm to a class of randomly generated instances.

Li and Lim (2001) proposed a Simulated Annealing (SA) and Tabu Search (TS) hybrid algorithm for multiple-vehicle VRPPDTW. Three neighborhoods are used for local search. Then a SA algorithm based on restart strategy is performed with tabu list to avoid cycling in the search process.

Lao and Liang (2002) described a TS for multiple-vehicle VRPPDTW. Before applying the TS, they obtain an initial solution by using a hybrid of standard insertion procedure and sweep procedure. The algorithm was tested on the Solomon's benchmark problems.

Dessouky et al. (2003) added an environmental component to the criteria function of an insertion algorithm for the VRPPDTW problem in addition to the standard cost component.

Lu et al. (2003) developed a new-insertion based construction heuristic for the multiple-vehicle VRPPDTW. While classical insertion methods select the feasible solution with least-increase in travel distance or time for next insertion, their method also takes into account the reduction in the slack which is time difference between the time window and the service time. They also described a measure called crossing length percentage to evaluate the visual attractiveness for the existing routes. They compared their heuristic with parallel and sequential insertion heuristics on different benchmark problems.

Bent and Hentenryck (2006) presented a two-stage hybrid algorithm for multi-vehicle VRPPDTW. The SA algorithm is used in the first stage to decrease the number of routes. In the second stage, Large Neighborhood Search is used to decrease total travel cost.

The studies related to the second class are as follows:

Jaw et al. (1986) proposed a heuristic algorithm based on sequential insertion for multiple-vehicle DARP with time windows. The heuristic selects the customers in the order of the increase of the earliest pickup time. They applied their heuristic to a real-life application with 260 customers and 20 vehicles.

Bodin and Sexton (1986) proposed a two-phase approach for multiple-vehicle DARP with time windows. Firstly, customers are clustered by a sequential insertion heuristic, and then routes are constructed.

Madsen et al. (1995) studied dynamic DARP with multiple vehicle capacities and multiple objectives. It is based on the insertion heuristic proposed by Jam et al (1986) and was tested on a real-life instance containing 24 vehicles and 300 users.

Cordeau and Laporte (2003) developed a tabu-search algorithm for multiple-vehicle DARP with time windows. They proposed a procedure for neighborhood evaluation to minimize route duration and ride times. They used randomly generated and real-life datasets.

Diana and Dessouky (2004) presented a parallel regret insertion heuristic to solve large scale DARPs with time windows. They tested their heuristic on data sets of 500 and 100 requests built from data of para-transit service in the Los Angeles County.

Melachrinoudis et al. (2005) proposed a TS algorithm for a double request DARP with soft windows and applied to the CAB Health and Recovery Services. They defined a multi-objective function which minimizes transportation cost and customer

inconvenience time.

Mauri and Lorena (2006) developed a SA algorithm for multiple-vehicle DARP with time windows. They described a multi-objective model and they performed several experiments with instances presented by Cordeau and Laporte (2003).

3. PROBLEM DESCRIPTION

In this chapter, the details of the problem and the mathematical model are provided. The inputs and outputs of the model, main assumptions, parameters, decision variables and possible objective functions are described. Lastly, the mathematical model formulation is given.

3.1. Model Formulation

In this problem, a heterogeneous vehicle fleet with different capacity, speed and cost characteristics must satisfy the transportation of a set of customer requests within time intervals of associated nodes. Each request is defined by a pickup node, a corresponding delivery node and a demand request to be transported between these nodes. The requested transport could involve goods or people. This latter environment is called a dial-a-ride problem. The objective function(s) generally minimizes the total system cost.

In the problems of this class, each node has a time interval (e, l) to be served in which e is the earliest service time and l is the latest service time for that node and the service at each node must start within the associated time window and the vehicle must stay at the node during the service. In order to guarantee feasibility during the progress of the algorithm, time window constraints are treated as soft constraints and it is permitted that time window constraint be violated at a cost. Otherwise, a vehicle is not allowed to serve a node outside the permissible time interval.

In addition to the time window constraints to be satisfied at each node, there are several other sets of constraints. These can be shortly summarized as visiting each pickup and delivery node exactly once, not exceeding the capacity of vehicles and coupling the pickup and corresponding delivery nodes on the same vehicle routes and visit precedence among each pickup node and its associated delivery node. There are also depot constraints that enable vehicles to return to the appropriate depots and

resource restrictions on the number of drivers and vehicle types.

3.1.1. Inputs of the Model

The inputs of the model are:

- the locations of pickup nodes
- the locations of delivery nodes
- customer transportation requirements
- vehicles which transport the demands between pickup and delivery nodes
- the origin and destination depot of vehicles
- capacity of vehicles
- service times at nodes
- traveling times
- distances between nodes
- time window intervals of all nodes.

3.1.2. Main Assumptions

A model is designed to perform the following conditions as described by Derigs and Döhmer (2006) :

- Each request is assigned to exactly one vehicle/route
- For each request, the pickup node is visited before the delivery node.
- Each customer is visited in its time interval.
- For each route, the load of the vehicle must not exceed the capacity of vehicle at any time
- All routes have the same origin and destination depot and each route has to start at the origin depot and to end at the destination depot. The origin and destination depots are denoted by ‘0’ and ‘ $2n + 1$ ’ respectively. (This condition is relaxed to include multiple depots in the SA algorithm.)
- A pre-determined number of vehicles is available.

- Vehicles differ in capacity but not speed, so the traveling time between customers does not depend on vehicles.
- The cost of vehicles or drivers is not considered.
- The time windows of nodes are strict. In the sense that a solution becomes infeasible if a node is supplied beyond the upper bound of its time window. In the case of soft time windows a later service does not affect the feasibility of the solution but is penalized by adding a penalty to the objective function. (Gribkovskaia, 2007).
- Distances between nodes are symmetric.
- The model is static and deterministic where the values of parameters are known with certainty when decisions are being made.
- The objective function of the problem is to minimize the total transportation cost in the system.

3.1.3. Outputs of the Model

The solution of the model will provide the route for each vehicle, time when each node is visited and the load profile of each vehicle at any time so as to satisfy the main assumptions described in Section 3.1.2.

3.1.4. Alternative Objective Functions

In this thesis, the objective function is chosen as the minimization of the traveling cost. However, without loss of generality, other objective functions could be chosen from among

- minimizing the number of vehicles required to serve all the customers
- balancing the route travel time and vehicle load
- minimizing the penalties associated with partial service of the customers
- minimizing the penalties for latest services exceeding the latest service time of customers
- or any combination of these objectives.

3.1.5. Parameters

i, j	: Indices for nodes	
k	: Index for vehicles	
K	: Set of vehicles	$k = \{1, 2, \dots, K\}$
n	: Number of customer requests	
P	: Set of pickup nodes	$= \{1, 2, \dots, n\}$
D	: Set of delivery nodes	$= \{n + 1, n + 2, \dots, 2n\}$
O	: Depot	$= \{0, 2n + 1\}$
N	: Set of nodes	$N = \{P \cup D \cup O\}$
C_k	: Capacity of vehicle k	$k \in K$
s_i	: Service time at customer i	$i \in N$
t_{ij}	: Travel time between nodes i and j	$i, j \in N$
c_{ij}	: Distance between nodes i and j	$i, j \in N$
a_i	: Earliest start time of service at node i	$i \in N$
b_i	: Latest start time of service at node i	$i \in N$
l_i	: The demand of customer i (i.e. pickup node i)	$l_i = -l_{n+i} \quad i \in P$
M	: A large number	

For depots, service times and requirements are assumed to be zero, that is,

$$l_0 = l_{2n+1} = s_0 = s_{2n+1} = 0$$

3.1.6. Decision Variables

x_{ijk}	=	$\begin{cases} 1 & \text{If arc } (i, j) \text{ is traversed by vehicle } k \\ 0 & \text{otherwise} \end{cases}$	$i, j \in N, \quad k \in K$
T_{ik}	=	Time when vehicle k starts the service at node i ,	$i \in N, \quad k \in K$
L_{ik}	=	Load of vehicle k after the service at node i has been completed,	$i \in N, \quad k \in K$

3.2. Mathematical Formulation of the Model

A mixed integer mathematical model is formulated to solve the problem as follows:

$$\text{Min} \quad \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \quad (3.1)$$

Subject to

$$\sum_{k \in K} \sum_{\substack{j \in N \setminus \{2n+1\} \\ j \neq i}} x_{ijk} = 1 \quad \forall i \in P \quad (3.2)$$

$$\sum_{k \in K} \sum_{\substack{j \in N \setminus \{0\} \\ j \neq i}} x_{ijk} = 1 \quad \forall i \in D \quad (3.3)$$

$$\sum_{\substack{j \in N \setminus \{0\} \\ j \neq i}} x_{ijk} - \sum_{\substack{j \in N \setminus \{2n+1\} \\ j \neq i}} x_{jik} = 0 \quad \forall k \in K, \forall i \in N \setminus \{0, 2n+1\} \quad (3.4)$$

$$\sum_{j \in P} x_{0jk} \leq 1 \quad \forall k \in K \quad (3.5)$$

$$\sum_{i \in D} x_{i,2n+1,k} \leq 1 \quad \forall k \in K \quad (3.6)$$

$$\sum_{\substack{j \in N \setminus \{0,2n+1\} \\ i \neq j}} x_{ijk} - \sum_{\substack{j \in N \setminus \{0,2n+1\} \\ j \neq i+n}} x_{j,i+n,k} = 0 \quad \forall k \in K, \forall i \in P \quad (3.7)$$

$$L_{0k} = 0 \quad \forall k \in K \quad (3.8)$$

$$L_{2n+1,k} = 0 \quad \forall k \in K \quad (3.9)$$

$$l_i \leq L_{ik} \leq C_k \quad \forall k \in K, \forall i \in P \quad (3.10)$$

$$L_{ik} \leq C_k - l_{i-n} \quad \forall k \in K, \forall i \in D \quad (3.11)$$

$$\begin{aligned} L_{ik} + l_j - L_{jk} &= (1 - x_{ijk}) M \quad \forall k \in K, \forall i \in N \setminus \{0, 2n+1\} \\ &\quad \forall j \in N \setminus \{0, i\} \end{aligned} \quad (3.12)$$

$$L_{0k} + l_j - L_{jk} = (1 - x_{0jk}) M \quad \forall k \in K, \forall j \in P \quad (3.13)$$

$$\begin{aligned} L_{jk} &\leq \sum_{\substack{i \in N \setminus \{2n+1\} \\ i \neq j}} x_{ijk} M \quad \forall k \in K, \forall i \in N \setminus \{2n+1\} \\ &\quad \forall j \in N \setminus \{0\} \end{aligned} \quad (3.14)$$

$$a_i \leq T_{ik} \leq b_i \quad \forall k \in K, \forall i \in N \quad (3.15)$$

$$T_{ik} + t_{i,i+n} x_{ijk} \leq T_{i+n,k} \quad \forall k \in K, \forall i, j \in P \quad (3.16)$$

$$\begin{aligned} T_{ik} + s_i + t_{ij} - T_{jk} &\leq (1 - x_{ijk}) M \quad \forall k \in K, \forall i \in N \setminus \{2n+1\} \\ &\quad \forall j \in N \setminus \{0\} \end{aligned} \quad (3.17)$$

$$T_{jk} \leq M \sum_{\substack{i \in N \setminus \{2n+1\} \\ i \neq j}} x_{ijk} \quad \forall k \in K, \forall j \in N \setminus \{0\} \quad (3.18)$$

$$x_{ijk} \in \{0, 1\} \quad \forall k \in K, \forall i, j \in N \quad (3.19)$$

$$T_{ik}, L_{ik} \geq 0 \quad \forall k \in K, \forall i \in N \quad (3.20)$$

Objective function 3.1 aims at minimizing the total transportation cost, while no other costs are considered. The constraints are categorized into 3 groups: routing-related constraints, load-oriented constraints and time window constraints. Constraints 3.2 - 3.6 are routing related constraints. Constraints 3.2 and 3.3 ensure that each pickup/delivery node is visited exactly once. In other words, every pickup/delivery node belongs to one and only one route, respectively. Constraint 3.4 guarantees that vehicles do not reside at the nodes; that is, they visit, perform service and leave the node unless the node is the depot. Constraints 3.5 and 3.6 imply that each vehicle starts its route from its original depot and terminates its route at its destination depot respectively. Original depot and destination depot could be the same or not. The inequality implies that some vehicles may not be assigned to a route. This type of constraints 3.5 - 3.6 try to minimize the number of vehicles used as well. It indirectly contributes to the objective function which has no ability of reducing the number of vehicles. Constraint 3.7 stipulates that a request is served by the same vehicle. A vehicle should visit both pickup and delivery node of a request. Hereby the nodes of a request take place in the same route. Constraints 3.8 - 3.14 are load-oriented constraints. Constraints 3.8 and 3.9 state that every vehicle starts and terminates its route empty. Each vehicle should be empty before visiting its first pickup node on route (each vehicle starts its route by visiting a pickup node) and deliver all demands picked before reaching at its destination depot. In Constraints 3.10 - 3.11, the load of each vehicle can not exceed the capacity of vehicle, but should include the demand of the node served currently. In other words, after a vehicle visits a pickup node, it should load the demand of that pickup node and after visiting a delivery node, the vehicle should deliver the demand of that delivery node. Constraint 3.12 - 3.13 ensure the compatibility between routing and loading decisions and express the material balance in a compatible situation. Constraint 3.12 is obtained by the conversion of non linear equation into the linear form given by

$$x_{ijk}(L_{ik} + l_j - L_{jk}) = 0 \implies L_{ik} + l_j - L_{jk} = (1 - x_{ijk})M \quad \forall k \in K, (i, j) \in N$$

where M is a large number. Constraint 3.13 is a simplified version of Constraint 3.12 for the origin depot. Constraint 3.14 is a technical constraint which associates L_{jk}

and x_{ijk} ; that is, if vehicle k does not visit node j , then its load after node j should be zero. Constraints 3.15 - 3.18 are time window constraints. Constraint 3.15 sets the bounds of the time window. Namely, the vehicle should start its service at any node within the time window of that node. Constraint 3.16 forces a vehicle to visit the customer's pickup node before its delivery node so that the precedence relationship can be maintained. Constraint 3.17 and Constraint 3.18 ensure the compatibility between route and schedule. Constraint 3.17 is obtained via linearization given by

$$x_{ijk}(T_{ik} + s_i + t_{ij} - L_{jk}) = 0 \implies T_{ik} + s_i + t_{ij} - T_{jk} \leq (1 - x_{ijk}).M \quad (3.21)$$

$$\forall k \in K, (i, j) \in N$$

where M is a large number. Constraint 3.18 is a technical constraint which associates T_{jk} and x_{ijk} ; that is, if vehicle k does not visit node j , then its start time of service should be zero. Constraint 3.19 guarantees the binary integrality of variables x_{ijk} . Constraint 3.20 represents the non-negativity restriction for corresponding variables. (The interested reader should refer to similar mathematical models by Kirkpatrick et al. 1983, Melachrinoudis et al. 2005, Nagy and Salhi 2005).

3.3. Dimensionality of the Model

- Number of delivery nodes: n
- Number of pickup nodes: n
- Number of vehicles: K
- Number of constraints: $O(n^2k)$
- Number of binary variables: $4(2n + 1)^2k$
- Number of continuous variables: $4nk$

As can be seen easily, it is not possible to solve the large versions of this problem in a reasonable amount time.

4. THE SOLUTION APPROACH

The problem under consideration is a mixed integer linear programming problem which is shown to be NP-hard by previous researchers. It is believed that one may never find a solution technique that will guarantee optimal solutions to larger instances for such problems in polynomial time as a function of the problems size. In such situations, heuristic algorithms are developed to solve these problems. In this chapter, a brief overview of combinatorial problems will be provided first. Then a short description of classical heuristic algorithms and metaheuristics will be given. Finally, the details of the Simulated Annealing algorithm designed in this study will be explained.

4.1. Combinatorial Problems

Many researchers have studied different solution methods to find optimal solutions to problems which can be structured as a function of some decision variables under some constraints. Such problems can be modeled generally as follows:

$$\begin{aligned} &\text{Minimize} && f(x) \\ &\text{Subject to} && g_i(x) \geq b_i \quad i = 1, \dots, m; \\ &&& h_j(x) = c_j \quad j = 1, \dots, n; \end{aligned}$$

where x is a vector of decision variables and $g_i(\cdot)$, $h_j(\cdot)$, $f(\cdot)$ are general functions and the objective is to minimize $f(x)$.

In case $g_i(\cdot)$, $h_j(\cdot)$, $f(\cdot)$ are linear functions of decision variables and decision variables are continuous, the problem becomes a linear programming problem. In other cases where decision variables are allowed to be discrete, the problem converges to a combinatorial problem. The assignment problem, the 0 – 1 knapsack problem, the set covering problem and the vehicle routing problem are types of combinatorial problem.

Combinatorial optimization problems are normally easy to describe but difficult to solve. The theory of computational complexity was discovered by Cook (1971) and it attempts to categorize the computational requirement of algorithms and classify problems met in practice as easy or hard. A problem is called easy if an algorithm can be developed that solves to optimality every instance of the problem in a polynomial-time complexity bounded by the size of the problem instance. Such a polynomial-time algorithm is said to be efficient. A problem is called hard if efficient algorithms to solve it do not exist. The results of the computational complexity theory provided evidence that many combinatorial problems are hard and belong to the class of NP-complete (non-deterministic polynomial-time complete) problems, Garey and Johnson (1979). An optimal algorithm for a problem in this class would require a number of computational steps that grows exponentially with the problem size. Hence, only small sized instances can be solved. The common belief is that no efficient algorithm could possibly be found to solve these inherently hard problems. Thus, heuristics (or approximate algorithms) are considered to be the only practical tools for solving hard combinatorial optimization problems. (Osman et al., 1996).

4.2. Classical Heuristics for Solving Vehicle Routing Problems

For VRP, several types of heuristics have been proposed. These could be basically grouped into two classes as classical heuristics and metaheuristics. Classical heuristic methods aim at a relatively limited exploration of the search space, thereby producing acceptable-quality solutions in modest computing times. Moreover, they can be easily extended for variant constraints of different real-life applications. On the other hand, metaheuristics perform a deep exploration in the solution space by combining neighborhood search methods, memory structures and recombinations of solutions so that they obtain higher quality solutions than classical heuristics. This makes their extension to different contents difficult.

Classical heuristics for VRP are broadly classified into three types. These are constructive heuristics, two-phase heuristics and improvement heuristics. Additional information on classical heuristics for the VRP can be found in studies of Christofides,

Mingozi, and Toth (1979), Bodin et al. (1983), Golden and Assad (1988) and Fisher (1995).

4.2.1. Constructive Heuristics

Constructive heuristics gradually build a feasible solution. They do not encompass an improvement phase. They perform two main operations. One is to merge existing routes using a savings criterion, and the other is to assign vertices gradually to vehicle routes using an insertion cost.

Clarke and Wright algorithm (1964) is the most widely known heuristic based on savings criterion. When two routes $(0, \dots, i, 0)$ and $(0, j, \dots, 0)$ can feasibly be merged into a single route $(0, \dots, i, j, \dots, 0)$, a distance savings $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ is generated. Since it tends to produce good routes at the beginning but less interesting routes toward the end, and it is time consuming, storing and sorting all savings, several enhancements have been made on Clarke and Wright algorithm. The interested reader can refer to Gaskell (1967), Yellow (1970) and Altinel and Öncan (2005) for enhancements.

The routes are constructed in a way such that the route starts with a tour on small subsets of nodes and then this tour is extended by inserting the remaining nodes one after the other until all nodes have been inserted. Generally a new node is inserted into the tour at the point that causes the minimum increase in the cost.

Mole and Jameson (1976) proposed an algorithm which expands one route at a time in which initially the number of vehicles is not specified. They use two parameters λ and μ to expand routes under construction:

$$\begin{aligned}\alpha(i, k, j) &= c_{ik} + c_{kj} - \lambda c_{ij}, \\ \beta(i, k, j) &= \mu c_{0k} - \alpha(i, k, j).\end{aligned}$$

After a route is constructed, the route is optimized by applying a 3-opt procedure.

4.2.2. Two-phase heuristics

Another proposed insertion heuristic for VRP is two-phase insertion heuristic by Christofides, Mingozzi, and Toth (1979). It firstly performs a sequential insertion heuristic and then a parallel route construction is performed based on the routes obtained in the first phase. It also uses a 3-opt procedure after a route construction.

In two-phase heuristics, the problem is decomposed into its two natural components, clustering of vertices into feasible routes and actual route construction.

Two-phase heuristics could be categorized into two groups: cluster-first, route-second methods and route-first, cluster-second methods.

In cluster-first, route-second heuristic, vertices are first grouped feasibly and a route is constructed for each of them. Some proposed algorithms of this type of heuristic are as follows:

Sweep Algorithm: Feasible clusters are initially formed by rotating a ray centered at the depot. A vehicle route is then obtained for each cluster by solving a TSP. (Gillett and Miller (1974)).

Fisher and Jaikumar Algorithm: It performs a Generalized Assignment Problem to cluster the vertices and then solves a TSP for each cluster to construct a route. (Fisher and Jaikumar (1981)).

Bramel and Simichi-Levi Algorithm: First, seeds of routes are determined by solving a capacitated location problem and the remaining vertices are inserted to their allotted routes with the least insertion cost. (Bramel and Simichi Levi (1995)).

Route first-cluster-second methods construct a TSP tour on all vertices, disregarding side constraints, and decompose this tour into feasible vehicle routes in a second phase. (Beasley (1983)).

4.2.3. Improvement Heuristics

Improvement heuristics perform on each vehicle route taken separately or on several routes at a time. The first case could be considered as a TSP problem and any improvement heuristic for TSP can be applied. Lins's (1965) λ -opt mechanism and Or's (1976) OR-opt method can be cited among many.

Van Breedam (1994) classified multiroute improvements as string cross, string exchange, string relocation, and string mix. (Thompson and Psaraftis (1993), and Kindervater and Savelsbergh (1997)).

4.3. Motivation for MetaHeuristic Approaches

Metaheuristics are a class of approximate methods that are designed to attack hard combinatorial optimization problems where classical heuristics have failed to be effective and efficient. Metaheuristics provide general frameworks that allow for creating new hybrids by combining different concepts derived from: classical heuristics; artificial intelligence; biological evolution; neural systems and statistical mechanics. These families of approaches include genetic algorithms, greedy random adaptive search procedure, problem-space search, neural networks, simulated annealing, tabu search, threshold algorithms and their hybrids.

A definition of meta-heuristics was made by Reeves (1993) as follows: "A heuristic is a technique which seeks good solutions (near optimal) at a reasonable computational cost without being able to guarantee either feasibility or optimality or even in many cases to state how close to optimality a particular solution is."

Heuristic methods were improved since it was observed that exact algorithms were capable of solving small instances of a problem, and they were not able to find optimal solutions to large instances in a reasonable amount of computing time.

The well known first popular heuristic is the local neighborhood search. It is very

simple and it results in relatively short computing time as defined in Table 4.1 .

Table 4.1. The pseudocode of the local neighborhood search

Choose $x \in X$ to start the process

Repeat Find $x' \in N(x)$ such that $f(x') < f(x)$

Replace x' by x

Until $f(x') > f(x)$ for all $x' \in N(x)$

where x is the local optimum and the method stops.

Such a strategy often results in convergence to a local rather than a global optimum and the local optimum produced by any local search descent procedure can be very far from optimality as simply illustrated in Figure 4.1. Local search methods perform a blind search since they only accept sequential solutions which produce reductions in the objective function value. They do not make use of any information gathered during the execution of the algorithm, and their performance depends heavily on initial solutions and the neighborhood generation mechanisms. In order to avoid some of the local search descent disadvantages, researchers have proposed different methods. Some has enlarged the neighborhood by moving from a 2-opt local search to a 3-opt one. Some has proposed to start the whole procedure from different initial solutions in the hope that it will converge to different local optima.

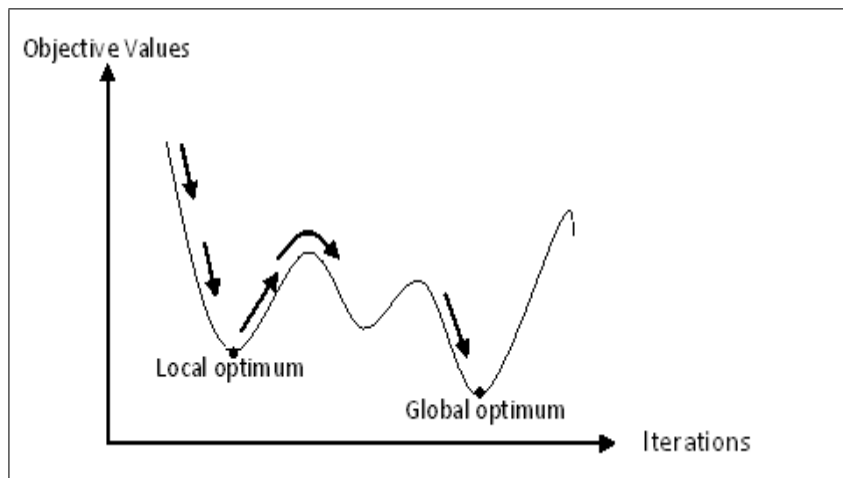


Figure 4.1. A conceptual representation of global and local optimum

4.4. Overview of General Simulated Annealing Metaheuristic Structure

SA principle is based on the annealing processing of the solids and it was firstly published by Metropolis (1953) in an algorithm to simulate the cooling of material in a heat bath. If solid material is heated past its melting node and then cooled back into a solid state, the structural properties of the cooled solid depend on the rate of cooling.

Contrary to well known local neighborhood search heuristic, SA allows uphill moves. Uphill moves consist of attempting to a neighborhood of the current solution which does not make any improvement on the current solution, but their frequency is managed by a probability function. The basis of using a probability function was also Metropolis' work in statistical thermodynamics. The laws of thermodynamics state that at temperature t , the probability of an increase in energy of magnitude E is given by $P(\delta E) = \exp(-\delta E/kt)$ where k is a physical constant known as Boltzman's constant. Here the temperature is decreased until the system freezes into a steady state. Kirkpatrick et al. (1983) and Cerny (1985) proposed to apply SA algorithms to combinatorial optimization problems. Basically the algorithm can be summarized in Table 4.2 .

Once the algorithm terminates, s_{best} is the approximation to the optimal solution. (Reeves et al., 1993) where s_0 is a solution in the solution space S , f is the objective function and N is neighborhood structure and it is a minimization problem. At the beginning of the algorithm, an initial solution s_0 , initial temperature t_0 , temperature reduction function α , acceptance probability function p and neighborhood structure N are determined and the algorithm starts by setting s_0 and t_0 to the current solution and current temperature, respectively. Also initially best solution is the initial solution, namely s_0 is assigned to s_{best} at the beginning. A neighborhood solution s of s_0 is found by using neighborhood structure and its f value based on the objective function is compared with that of s_0 . In case it is less, it is moved to s and s becomes the current solution. Also if s is less than s_{best} , s becomes s_{best} which is the best solution found up to that time. In other cases, s is not rejected immediately since SA lets uphill moves which have no improvement on the objective function. A number x is generated

Table 4.2. General structure of the SA algorithms

```

Select an initial solution  $s_0$ ;
Select an initial temperature  $t_0 > 0$ ;
Select a temperature reduction function  $\alpha$ ;
Assign  $s_{best} = s_0$ ;
Repeat
  Repeat
    Randomly select  $s \in N(s_0)$ ;
     $\Delta = f(s) - f(s_0)$ ;
    If  $\Delta < 0$  then  $s_0 = s$ ;
    If  $s_0 < s_{best}$  then  $s_{best} = s_0$ ;
  else
    generate random  $x$  uniformly in the range(0, 1);
    if  $x < \text{acceptance probability function } p$  then  $s_0 = s$ ;
  Until  $iteration\_count = nrep$ ;
  Set  $t = \alpha(t)$ ;
Until stopping condition = true.

```

randomly in uniformly distributed range of $(0, 1)$ and is checked with p . If it is smaller than p , s becomes the current solution. At each iteration, t is reduced according to function $\alpha(t)$. The algorithm stops when the stopping condition is reached, and s_{best} is recorded as the result.

In the SA algorithm, two types of decisions are made. One type is called generic decisions and they are related to parameters of the algorithm itself. These are the initial temperature, the cooling Schedule with respect to $nrep$ and temperature reduction function α and the stopping condition. The second type consists of problem specific decisions and involves the choice of the space of feasible solutions, the cost function and the neighborhood structure. The algorithmic design decisions for the SA metaheuristic procedure developed in this thesis are described in the next section.

4.5. Description of the SA Algorithm

The procedures for finding the initial solution and generating the neighboring solutions are the most important aspect of any SA algorithm. Furthermore, the performance of the algorithm highly depends upon the right choices of the acceptance probability function, the cooling function and the initial values of all these parameters. These algorithmic properties are described in detail.

Initial Temperature: The analysis of previous studies on cooling process has showed that main progress is achieved in the middle of the process for most problems. Hence, some researchers claimed that much time should not be spent at the beginning so that cooling should be done rapidly during this phase. Others suggested to start the process at lower temperatures. This idea can result in a convergence to a descent algorithm provided that the initial temperature is not high enough, since at lower temperatures little movement is possible and few increases in the objective are accepted.

Here, one of the pre-assigned vehicles in the problem is randomly selected and that vehicle serves all requests which are randomly ordered. While it serves a request, it visits pickup and delivery nodes of that request consecutively. In this way, the precedence relationship is maintained but the other constraints are not checked. This procedure continues for a specific iteration number k and at each iteration, the cost function of route r is calculated as $f(r_i)$.

The value of the initial temperature T_{begin} is determined by

$$T_{begin} = \frac{\sum_{i=1}^k f(r_i)}{k} \quad (4.1)$$

T_{begin} is the average of cost functions obtained at each iteration. This way the initial temperature is made problem-dependent.

Acceptance probability function: In the SA algorithm, uphill moves are allowed

by this function. The frequency of uphill moves is different at each level of temperature. Generally the number of uphill moves decreases when the temperature gets lower. The principle of this function enables the algorithm not to get stuck in local optima and to find global optima.

The basis of acceptance probability was inspired from Boltzman distribution, $p(\delta)=exp(-\delta/t)$. Different acceptance probabilities have been experimented, since it is well-known that the standard exponential distribution does not perform well for particular problem, and it decreases the speed of algorithm.

Johnson et al.(1983) found that the calculation of $exp(-\delta/t)$ takes one third of the total computation time in a SA algorithm in a graph partitioning problem. They suggest two methods of improvement. The first is to use $p(\delta) = 1 - \delta/t$ which approximates the exponential, and the second and better approach is to use a discrete approximation δ/t .

In this study, 4 types of acceptance probability function have been used. Three of them are functions mentioned above and the last one was proposed by Ozdamar et al. (2001).

$$P_a = e^{-\delta/T_r} \quad (4.2)$$

$$P_a = \delta/T_r \quad (4.3)$$

$$P_a = 1 - \delta/T_r \quad (4.4)$$

$$P_a = e^{-\delta/(T_r*f_{current})} \quad (4.5)$$

where T_r is the current temperature, δ is the difference between the objective function values of neighborhood solution and current solution, and $f_{current}$ is the cost of current solution.

The cooling function: As it is well-known, the initial temperature is reduced by the cooling function. In this thesis, 2 types of cooling function have been applied. The

first one is given by

$$T_r = \alpha T_{r-1} \quad (4.6)$$

where T_r is current temperature and $\alpha < 1$. It is simply a geometric function. In the literature it is observed that higher α values give better results especially when α is between 0.8 and 0.99.

The second one, on the other hand, is

$$T_r = \frac{T_{r-1}}{1 + \alpha\sqrt{T_{r-1}}} \quad (4.7)$$

where T_r is the current temperature at iteration r and $\alpha < 1$ a small time constant. It is a simple monotonically decreasing function. The square root of T_{r-1} is introduced in the denominator to speed the cooling process. (Tan et al., 2001)

Temperature iteration function: As mentioned earlier, it is better not to spend so much time at the beginning of the procedure, that is, in higher temperatures it is better to increase the number of iteration when T_r value decreases. Thus this approach enables to spend longer time at lower temperatures as we get closer to the optimum. The function used in this study is given by

$$iter_{T_r} = \beta iter_{T_{r-1}} \quad (4.8)$$

where $iter_{T_r}$ is the iteration number of current temperature and β is a constant greater than 1.

Initial Solution: Most heuristic search strategies involve finding an initial feasible solution and then improving on that solution using local or global optimization techniques. Here, we use a two-step method to create an initial solution.

At the first step, it is assumed that initially the number of vehicles is equal to the

number of the requests. Each request which has both a pickup and a delivery node is assigned to a vehicle randomly. For each assignment, all feasibility conditions including time windows, load capacity and precedence are checked and feasible assignments are accepted.

At the second step, the solution obtained at the first step is improved by merge operations. If the number of routes decreases after a feasible merge move, the solution before the reduction is stored and this step is repeated until a predetermined number of successive infeasible merge moves is rejected.

At the end of this step, there is a solution pool which includes the solutions stored and each solution in the solution pool is a candidate to be an initial solution for the SA algorithm.

The aim of storing solution is that the vehicles which are excluded after a merge move are not used again. At the same time, different initial solutions in the SA increases diversification.

Types of Moves: A move can be defined as the change of nodes between routes or within routes in order to obtain a neighborhood solution from the current solution. It is assumed that only feasible moves with respect to the time window, load capacity and precedence constraints are performed. In this thesis, two types of moves are used, namely the Request Exchange and Merge by convention.

Request Exchange generates a neighborhood solution by exchanging pickup and delivery nodes of two requests respectively. This move could be used both when two chosen requests are on same route or on different routes. Symbolically, this can be represented by $R_1(P_1, D_1)$ and $R_2(P_2, D_2)$ where R_1 and R_2 are requests which have a pickup and a delivery node to be exchanged. P_1 replaces with P_2 and D_1 replaces with D_2 at the same time.

An exchange operation performed on two different routes is illustrated in Figure

4.2. A similar operation can be performed between the two arcs on the same route.

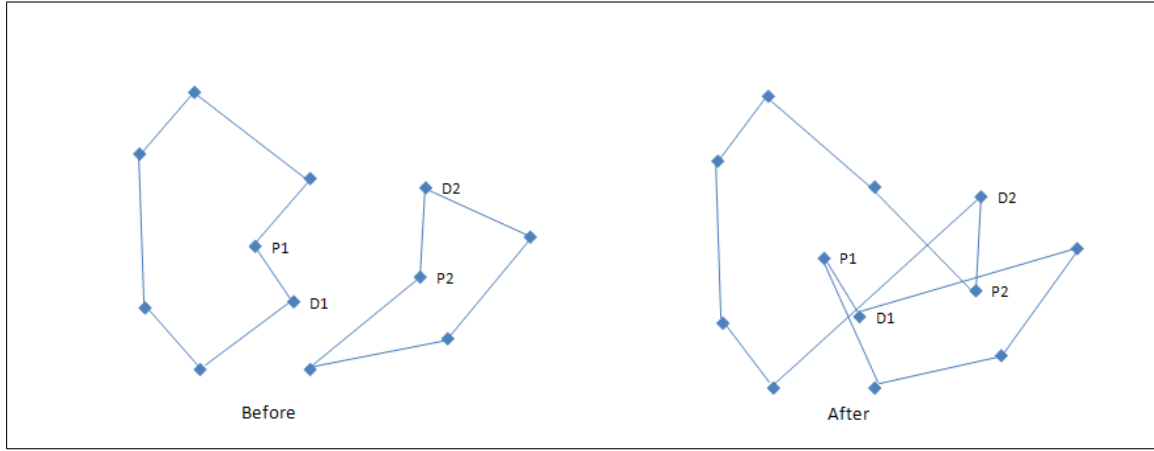


Figure 4.2. Move type of request exchange

Merge move can be described as the move of a request from one route to another. Merge move aims to reduce the number of routes. Two randomly chosen requests should be on different routes for Merge move. While Merge move tries to relocate each chosen request to other one's route, it stores all possible feasible moves and performs the one with the lowest cost.

A merge operation which requires two different routes is illustrated in Figure 4.3. A merge operation is performed on two chosen requests $R_1(P_1, D_1)$ and $R_2(P_2, D_2)$. After merge, the request R_2 locates on the route of request R_1 .

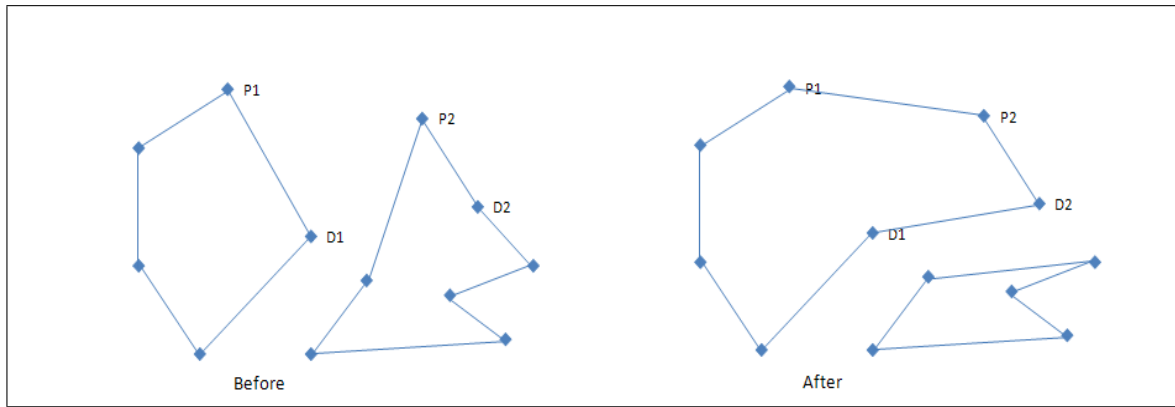


Figure 4.3. Move type of merge

Termination Criterion: The termination condition of the SA algorithm is based on

two criteria. One of them is a number of iterations $max_{iter_algorithm}$ assigned initially for the whole procedure, and the second is a predetermined threshold temperature $T_{threshold}$. If the current iteration count of the algorithm is greater than $max_{iter_algorithm}$ or if the current temperature is less than $T_{threshold}$, the algorithm stops. The pseudocode of the algorithm is given in Table 4.3.

Table 4.3. The pseudocode of the SA algorithm

```

GENERATE initial solution resulting in a pool of initial solutions  $SP = \{s_1, s_2, \dots, s_n\}$ ;
GENERATE initial temperature  $T_{begin}$  ;
SELECT a cooling function  $T_r$ ;
SELECT an acceptance probability function  $P_a$ ;
SELECT a temperature iteration function  $Iter_{T_r}$  ;
ASSIGN  $s_0 = s_1$ ; and  $s_{best} = s_1$  where  $s_1 \in SP$ ;
ASSIGN  $iter_{algorithm} = 0$ ;  $iter_{temp} = 0$ ;
FOR ( each solution in the solution pool  $SP$ )
    ASSIGN  $S_{current} = s \in SP$ ;
    SELECT  $\max_{iter_{temp}}$ ;
    ASSIGN  $iter_{algorithm} = 0$ ;  $iter_{temp} = 0$ ;  $T_{current} = T_{begin}$ ;
    IF  $f(S_{current}) < f(S_{best})$  THEN
         $S_{best} = S_{current}$ ;
    FOR ( $algorithm\_iteration < \text{maximum number of iteration for algorithm}$  ||  $T_{current} > T_{threshold}$ )
        FOR ( $temperature\_iteration < \text{maximum number of iteration for temperature}$ )
            SELECT two requests randomly;
            IF chosen requests are on the same route THEN
                GENERATE neighborhood solution by Request Exchange;
            ELSE {
                GENERATE neighborhood solution by Merge;
                IF Merge move does not result in any feasible solution THEN
                    GENERATE neighborhood solution by Request Exchange;}
            LIST the feasible neighborhood solutions  $N(S) = \{N_{s_1}, N_{s_2}, \dots\}$  where  $f(N_{s_i}) \leq f(N_{s_j})$   $i < j$ ;
            IF  $f(N_{s_1}) < f(S_{best})$  THEN {
                 $S_{best} = N_{s_1}$ ;  $S_{current} = N_{s_1}$ ; }
            ELSE {
                 $\Delta = f(N_{s_1}) - f(S_{current})$ ;
                IF  $\Delta < 0$  THEN
                     $S_{current} = N_{s_1}$ ;
                ELSE IF  $random([0, 1]) \leq P_a$  THEN
                     $S_{current} = N_{s_1}$ ; }
            UPDATE  $T_{current}$  by  $T_r$ ;
            UPDATE  $\max_{iter_{temp}}$  by  $Iter_{T_r}$ ;
        END-FOR ;
    END-FOR;
END-FOR;

```

The detailed description is given below:

- 1: A pool of initial solutions are generated by the initial solution method.
- 2: Each solution in the solution pool is chosen as the current solution in the SA sequentially. If there is no remaining solution in the initial solution pool which is not used in SA, then the algorithm is terminated.
- 3: The current temperature $T_{current}$ is set equal to the initial temperature T_{begin} .
- 4: Two requests among all requests are chosen randomly.
- 5: The routes of chosen requests are checked whether they are served by the same vehicle or not, in other words whether they are on the same route or not. If they are on the same route, one Request Exchange move is applied. If they are not, one Merge move is operated. When the Merge move does not result in any feasible solution, a Request Exchange move is applied. As a result neighborhood solutions of the current solution are obtained and only feasible ones are stored.
- 6: The objective function value (total travel time) f is computed for each feasible neighborhood solution, and the neighborhood solution with the least cost is selected.
- 7: The difference between the objective function value of the neighborhood solution and that of the current solution Δ is computed. If $\Delta > 0$ and $r \geq P_a$ where r is a pseudo-random number in $[0, 1]$, then proceed to 4. Otherwise, continue.
- 8: The neighborhood solution becomes the current solution and is saved if it is the best solution found so far.
- 9: Two controls are executed with respect to the iteration number. One is to check the iteration number of the current temperature $iter_{temp}$ and the other is to check the iteration number for the whole algorithm $iter_{algorithm}$. Naturally, $iter_{algorithm}$

increases by one at each iteration during the algorithm; $iter_{temp}$ also increases by one each iteration but it is set to 0 at each time that temperature is decreased by cooling function. These maximum iteration numbers are preassigned at the beginning of the algorithm as max_{iter_temp} and $max_{iter_algorithm}$ respectively. Although $max_{iter_algorithm}$ is constant, max_{iter_temp} changes at each temperature reduction by the temperature iteration function $iter_{Tr}$.

If $iter_{temp}$ is greater than max_{iter_temp} , the current temperature $T_{current}$ decreases by the cooling function T_r and max_{iter_temp} is updated by the temperature iteration function for the next temperature and $iter_{temp}$ is set to 0. Then it is checked whether $iter_{algorithm}$ is greater than the $max_{iter_algorithm}$ or $T_{current}$ is smaller than $T_{threshold}$. If either is true, it means that the algorithm ends for the chosen initial solution and another initial solution in the solution pool is selected to continue. In that case, proceed to 2. Otherwise go to 4.

10: The solution saved as the best solution is the final best solution of the problem under consideration.

5. EXPERIMENTAL DESIGN

In all experiments, a given set of values are assigned to the SA parameters as given below:

Cooling rate (α): It is the fraction used in the cooling function. As mentioned earlier, 2 types of cooling function have been used. α takes the value of 0.9 and 0.1 in equation (4.6) and in equation (4.7) respectively.

Temperature iteration rate (β): This is the fraction in the temperature iteration function (4.8) which increases the iteration number of temperature each time temperature decreases and β is taken as 1.1.

Threshold T: It is one of the stopping criteria for the algorithm and is denoted by $T_{threshold}$. When the current temperature gets lower than $T_{threshold}$, which is chosen as 250 the algorithm ends.

The SA algorithm has been tested for 4 types of acceptance probability functions and these tests are implemented on different problem sizes.

Four different probability functions that are tested in this study are given by:

1. $e^{-\Delta/f_{current}*T_r}$
2. e^{Δ/T_r}
3. $1 - \Delta/T_r$
4. Δ/T_r

Before solving the model with CPLEX and the SA algorithm, test problems have been generated. Since the aim of the experimental design is to test the performance of the proposed algorithm especially when the problem gets larger, a test-bed of problem instances is developed with a different number of nodes.

It is well-known that the model dimensionality in these problems is determined by the number of pick-up and delivery (PD) points, and without loss of generality, the number of PD points are chosen as $\{4, 5, 6, 10, 20, 50, 75, 100\}$.

10 problem instances are examined for problems 4 PD, 5 PD and 6 PD while five instances are examined for problems 10 PD, 20 PD and 50 PD. Only one instance is analyzed for 75 PD and 100 PD simply to observe if the SA heuristic algorithm is capable of solving large problems of that size.

The required parameters which are service times, (x, y) coordinates, demand requirements and vehicle capacities for all nodes of each test problem have been generated randomly over uniform distribution. Time windows of the nodes are adjusted manually so as to guarantee feasibility. For all instances, service time for pickup nodes and corresponding delivery nodes and demand requirements have been generated randomly in uniformly distributed ranges of $[0, 10]$ and $[0, 100]$, respectively. The earliest service time of all nodes are assumed to be 0.

In Table 5.1, the number of pickup and delivery nodes (NPD), the number of problem instance (NPI), range for speed of vehicles (that is sometimes a single value), range for coordinates of nodes (x, y) , latest time window (LTW) of nodes, latest time window (LTW) of vehicles and range for vehicle capacities (VC) are illustrated.

Table 5.1. Ranges of the parameter values for different problem sizes

NPD	NPI	Speed	(x, y)	LTW for Nodes	LTW for Vehicles	VC
4 PD	1	2	(0,100)	400	150	(100,200)
4 PD	2	2	(0,100)	400	150	(100,200)
4 PD	3	(1,2)	(0,100)	400	200	(100,200)
4 PD	4	(1,2)	(0,50)	400	100	(100,200)
4 PD	5	(1,2)	(0,50)	400	100	(100,200)
4 PD	6	1	(0,50)	400	150	(100,200)
4 PD	7	(1,2)	(0,200)	400	250	(100,200)
4 PD	8	1	(0,100)	400	300	(100,200)
4 PD	9	1	(0,100)	400	300	(100,200)
4 PD	10	1	(0,150)	400	350	(150,200)
5 PD	1	(1,2)	(0,100)	400	150	(100,200)
5 PD	2	(1,2)	(0,100)	400	250	(100,200)
5 PD	3	(1,2)	(0,100)	400	200	(100,150)
5 PD	4	(1,2)	(0,100)	400	250	(100,200)
5 PD	5	(1,2)	(0,200)	400	400	(150,200)
5 PD	6	(1,2)	(0,200)	400	400	(100,200)
5 PD	7	(1,2)	(0,100)	400	300	(100,200)
5 PD	8	(1,2)	(0,100)	400	200	(150,200)
5 PD	9	(1,2)	(0,100)	400	200	(150,200)
5 PD	10	(1,2)	(0,200)	400	400	(100,200)
6 PD	1	(1,2)	(0,100)	400	250	(100,200)
6 PD	2	(1,2)	(0,200)	400	300	(100,200)
6 PD	3	(1,2)	(0,200)	400	300	(100,200)
6 PD	4	(1,2)	(0-200)	400	400	(100,200)
6 PD	5	(1,2)	(0,50)	400	100	(100,200)
6 PD	6	(1,2)	(0,100)	400	200	(100,200)
6 PD	7	(1,2)	(0,200)	400	300	(100,200)
6 PD	8	(1,2)	(0,100)	400	300	(100,200)

Table 5.1 – (cont.)

NPD	NPI	Speed	(x, y)	LTW for Nodes	LTW for Vehicles	VC
6 PD	9	(1,2)	(0,100)	400	250	(100,200)
6 PD	10	(1,2)	(0,100)	400	250	(100,200)
10 PD	1	1	(0,50)	150	150	(100,150)
10 PD	2	1	(0,50)	150	150	(100,150)
10 PD	3	1	(0,100)	250	250	(100,150)
10 PD	4	1	(0,150)	400	400	(200,250)
10 PD	5	1	(0,150)	500	500	(200,250)
20 PD	1	1	(0,100)	600	600	(300,400)
20 PD	2	1	(0,100)	500	500	(200,300)
20 PD	3	1	(0,100)	400	400	(200,300)
20 PD	4	1	(0,150)	800	800	(200,300)
20 PD	5	1	(0,150)	800	800	(200,300)
50 PD	1	1	(0,100)	1000	1000	(300,400)
50 PD	2	1	(0,100)	1000	1000	(300,400)
50 PD	3	1	(0,100)	1000	1000	(200,300)
50 PD	4	1	(0,100)	1000	1000	(200,300)
50 PD	5	1	(0,100)	1000	1000	(300,400)
75 PD	1	1	(0,100)	1000	1000	(300,400)
100 PD	1	1	(0,200)	1500	1500	(400,500)

All the tests were performed in a desktop with Intel®Core™ 2 of 2.4 GHz processor and 4 GB RAM memory , and using the operator system Microsoft Windows XP Professional x 64 Edition. The complete implementation algorithm was developed in Microsoft Visual Studio C++, version 2005 and it was also run by CPLEX Solver of GAMS, version 2.0.13.0 to find the optimum solutions of the problems.

These generated test problems are solved by using both CPLEX and the SA

algorithm and the results have been compared.

As an illustrative example, the main steps of the algorithm are shown in detail for one particular instance of a 4 PD problem. Data for the problem under consideration is given in Tables 5.2- 5.3 in which *NV* refers to the number of vehicles available in the problem instance.

Table 5.2. Data set of the 4 PD problem

Node	x	y	Demand	Service Time	ETW	LTW
0	72.67	87.60	53.94	1.42	0	400
1	84.37	77.97	61.15	3.92	0	400
2	9.26	37.59	0.88	9.19	0	400
3	72.65	83.76	74.37	4.69	0	400
4	92.57	95.59	-53.94	1.42	0	400
5	99.97	99.68	-61.15	3.92	0	400
6	5.62	67.72	-0.88	9.19	0	400
7	20.54	48.49	-74.37	4.69	0	400

Table 5.3. Vehicle data set of the 4 PD problem

NV	VC	Speed	x	y
1	112.72	1.62	58.50	47.99
2	164.57	1.49	17.41	85.89
3	197.71	1.06	57.12	60.18
4	132.61	1.22	78.33	80.26

The solution of the problem is obtained at the end of 6 iterations. In the first iteration, an initial solution is obtained with an objective function value of 480.091. The last iteration is the final solution whose objective value is 245.14.

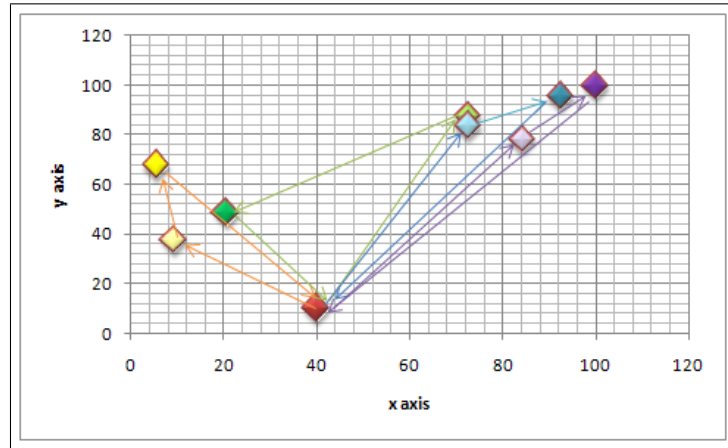


Figure 5.1. Iteration 1 of the solution approach

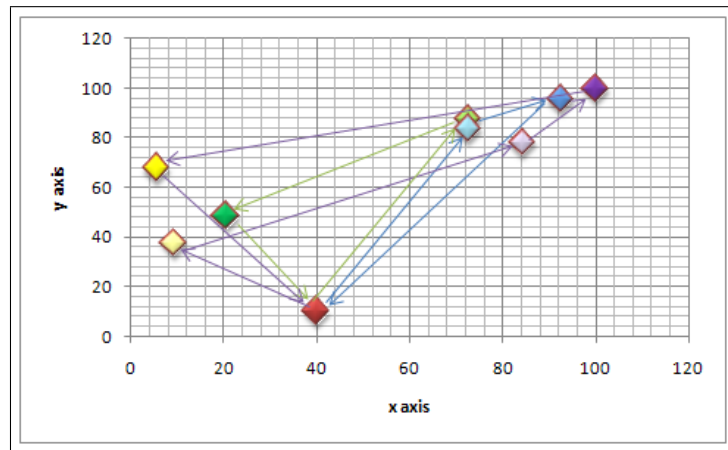


Figure 5.2. Iteration 2 of the solution approach

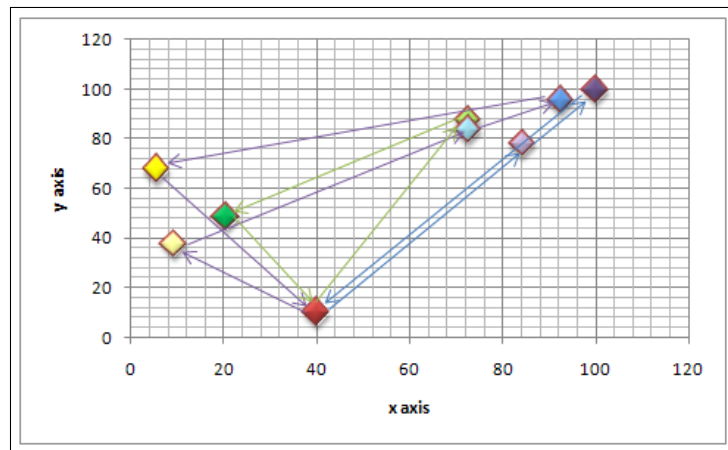


Figure 5.3. Iteration 3 of the solution approach

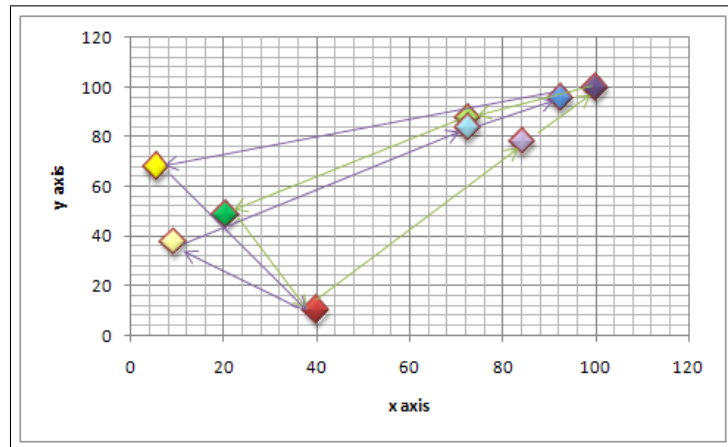


Figure 5.4. Iteration 4 of the solution approach

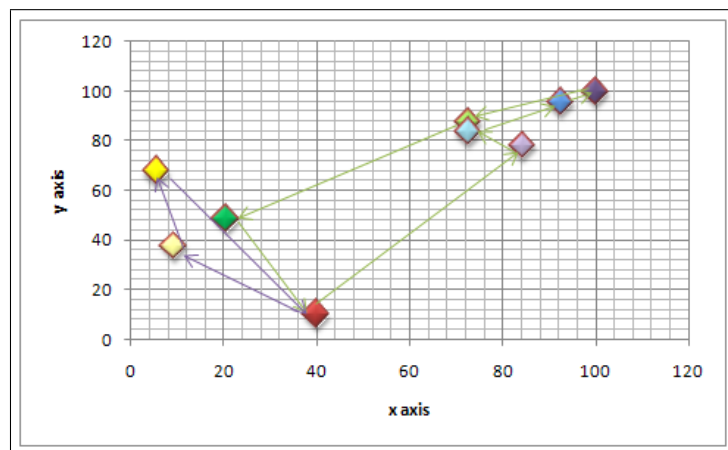


Figure 5.5. Iteration 5 of the solution approach

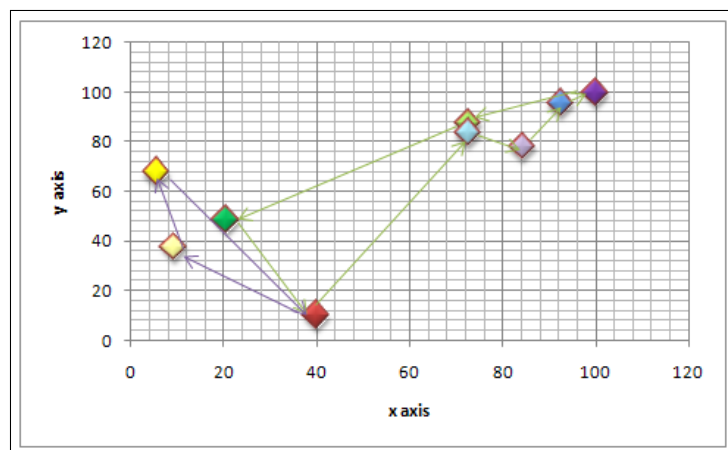


Figure 5.6. Iteration 6 of the solution approach

The heuristic results with different probability function of other problems are summarized in Tables 5.4 - 5.11. The route-based results are illustrated in Appendix A.1 - A.8. The CPLEX solutions could not be displayed for large problems when an optimum solution can not be found. In these tables, *BHS* denotes the best heuristic, *SOLN* denotes the optimum solution, f refers to objective function value, ET depicts elapsed time. Furthermore different probability functions are experimented, namely $P_1 = e^{-\Delta/f_{current} * T_r}$, $P_2 = e^{\Delta/T_r}$, $P_3 = 1 - \Delta/T_r$, $P_4 = \Delta/T_r$.

Table 5.4. Results of the SA heuristic with different probability functions for 4 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET(sec)	f	ET(sec)	f	ET(sec)	f	ET
1	232.14	22.13	232.14	21.36	232.14	21.08	232.14	22
2	201.16	21.72	201.16	19.765	201.16	19.92	201.16	21.31
3	245.14	7.64	245.14	6.843	245.14	6.766	370.40	6.89
4	164.91	21.38	164.91	20.703	164.91	20.77	164.91	21.34
5	166.15	6.49	166.15	6.188	166.15	6.25	166.15	6.33
6	318.99	6.89	318.99	6.609	318.99	6.56	318.99	6.81
7	559.55	6.98	559.55	6.984	559.55	6.84	559.55	7.02
8	475.87	20.91	475.87	20.11	475.87	20.11	475.87	21.22
9	430.02	21.31	430.02	21.51	430.02	21.55	430.02	21.11
10	624.20	12.88	624.20	12.33	624.20	12.56	624.20	12.78

Table 5.5. Results of the SA heuristic with different probability functions for 5 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET(sec)	f	ET(sec)	f	ET(sec)	f	ET(sec)
1	344.75	24.52	345.70	23.02	345.70	23.36	344.75	23.45
2	591.75	8.125	591.75	7.547	591.75	7.52	591.75	7.84
3	227.10	51.69	235.04	53.40	235.04	52.42	227.10	51.94
4	816.46	7.125	816.46	6.781	816.46	6.672	816.46	7.22
5	592.65	37.59	592.65	35.70	592.65	35.47	592.65	37.30
6	600.75	38.81	600.75	39.48	600.75	40.09	600.75	37.89
7	289.25	5.31	290.31	5.234	290.31	5.17	299.24	13.42
8	258.40	39.61	260.64	37.86	260.64	37.56	258.40	40.53
9	353.80	26.77	352.30	26.45	352.30	26.16	352.30	26.44
10	593.15	37.09	593.15	34.70	593.15	34.03	590.20	38.03

Table 5.6. Results of the SA heuristic with different probability functions for 6 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET(sec)	f	ET(sec)	f	ET(sec)	f	ET(sec)
1	328.32	51.44	352.38	50.36	328.32	50.72	353.64	51.53
2	758.92	36.97	758.92	36.03	758.92	36.31	758.92	37.25
3	765.15	27.80	770.39	27.19	841.98	26.31	772.62	27.95
4	697.65	32.45	696.29	32.64	696.29	32.69	675.09	33.84
5	185.55	33	175.12	33.08	175.12	33.11	188.12	33.88
6	340.35	56.75	340.35	55.20	340.35	54.14	340.35	55.36
7	847.17	24.63	847.17	25.02	847.17	25.69	847.17	25.13
8	347.38	57.17	383.49	52.63	383.49	48.34	347.38	58.78
9	371.89	57.59	367.74	56.36	367.74	56.38	367.74	57.66
10	331.72	60.28	332.9	57.58	344.08	59.97	331.72	60.13

Table 5.7. Results of the SA heuristic with different probability functions for 10 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET (sec)	f	ET (sec)	f	ET (sec)	f	ET (sec)
1	472.12	753.95	472.12	729.78	472.12	733.05	472.12	758.72
2	507.13	657.28	507.13	609.13	507.13	604.36	507.131	643.60
3	1027.02	543.48	1027.02	522.78	1027.02	523.88	1027.02	544.13
4	1395.80	632.41	1395.80	628.72	1395.80	629.63	1395.80	647.96
5	1131.57	1041.36	1131.57	1036.09	1131.57	1038.77	1131.57	1045.52

Table 5.8. Results of the SA heuristic with different probability functions for 20 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET (sec)	f	ET (sec)	f	ET (sec)	f	ET (sec)
1	957.66	4720.94	949.13	4767.58	936.64	4785.69	954.05	4707.37
2	1330.49	1838.30	1332.89	1954.99	1368.90	1816.14	1361.57	1683.42
3	1024.20	1688.87	1011.43	1724.31	1013.92	1704.26	1009.58	1639.76
4	1386.00	4076.74	1365.32	4147.53	1368.08	4121.23	1369.04	3976.04
5	2171.38	1324.38	1952.58	1384.35	2105.31	1307.84	2083.29	1319.62

Table 5.9. Results of the SA heuristic with different probability functions for 50 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET (sec)	f	ET (sec)	f	ET (sec)	f	ET (sec)
1	2304.20	25907.23	2245.68	26452.50	2275.49	26646.02	2393.65	25424.75
2	2088.87	30039.70	1919.04	33705.88	2119.98	32970.21	2102.62	29636.96
3	2388.65	23401.82	2348.68	26399.64	2343.19	26792.84	2463.22	25112.06
4	2560.60	11879.53	2518.62	11973.40	2518.62	11875.30	2512.99	11794.09
5	3696.91	5319.34	2550.59	14484.80	2615.95	14474.17	2631.32	14087.70

Table 5.10. Results of the SA heuristic with different probability functions for 75 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET (sec)	f	ET (sec)	f	ET (sec)	f	ET (sec)
1	3304.87	39392.81	3291.04	40533.42	3248.85	26552.92	3325.71	38791.07

Table 5.11. Results of the SA heuristic with different probability functions for 100 PD

	P₁		P₂		P₃		P₄	
NPI	f	ET (sec)	f	ET (sec)	f	ET (sec)	f	ET (sec)
1	8003.61	24027.86	7896.66	26552.92	7723.06	26507.65	8150.52	25437.82

The results are summarized in Table 5.12 to provide the objective function values and computational time for the SA algorithm and the CPLEX solution whenever possible.

Table 5.12. The summary of the results

			<i>CPLEX</i>		<i>BHS</i>	
NPD	NPI	NITER	f	ET(sec)	f	ET(sec)
4	1	5,000	232.135	134.266	232.135	21.078
	2	5,000	201.16	28.422	201.16	19.765
	3	5,000	245.14	10.967	245.14	6.766
	4	5,000	164.9065	62.727	164.914	20.703
	5	5,000	166.152	30.028	166.152	6.188
	6	5,000	318.99	139.875	318.99	6.562
	7	5,000	559.552	205.541	559.552	6.844
	8	5,000	475.871	60.431	475.871	20.109
	9	5,000	430.016	65.727	430.016	21.109
	10	5,000	624.198	41.198	624.198	12.328

Table 5.12 – (cont.)

			<i>CPLEX</i>		<i>BHS</i>	
NPD	NPI	NITER	f	ET(sec)	f	ET(sec)
5	1	5,000	344.749	11701.193	344.749	23.453
	2	5,000	591.747	30950.15	591.747	7.516
	3	5,000	227.095	287.907	227.095	51.685
	4	5,000	816.462	12917.625	816.462	6.672
	5	5,000	592.645	2399.455	592.645	35.467
	6	5,000	600.75	6058.14	600.75	37.891
	7	5,000	289.263	153.695	289.263	5.313
	8	5,000	258.3525	44.763	258.398	39.607
	9	5,000	352.2933	3387.687	352.295	26.156
	10	5,000	590.202	988.34	590.202	38.03
6	1	5,000	—	—	328,324	50,717
	2	5,000	—	—	758.921	36.031
	3	5,000	—	—	765.15	27.796
	4	5,000	673.6755	235132.438	675.089	33.844
	5	5,000	175.118	9339.693	175.118	33.078
	6	5,000	—	—	340,346	54.141
	7	5,000	—	—	847,17	24.625
	8	5,000	345.5915	50704.404	347.375	57.172
	9	5,000	367.739	68381.526	367.739	56.359
	10	5,000	327.1789	15924	331.715	60.125
10	1	100,000	-	-	472.119	729.782
	2	100,000	-	-	507.131	604.364
	3	100,000	-	-	1027.015	522.781
	4	100,000	-	-	1393.158	647.957
	5	100,000	-	-	1131.568	1036.093

Table 5.12 – (cont.)

			<i>CPLEX</i>		<i>BHS</i>	
NPD	NPI	NITER	f	ET(sec)	f	ET(sec)
20	1	100,000	-	-	936.635	4785.687
	2	100,000	-	-	1330.485	1838.297
	3	100,000	-	-	1011.434	1724.313
	4	100,000	-	-	1365.324	4147.532
	5	100,000	-	-	1952.581	1384.353
50	1	100,000	-	-	1952.581	1384.353
	2	100,000	-	-	1952.581	1384.353
	3	100,000	-	-	1952.581	1384.353
	4	100,000	-	-	2512.993	11794.088
	5	100,000	-	-	2550.585	14484.80
75	1	100,000	-	-	3248.853	26552.92
100	1	100,000	-	-	7723.063	26507.646

The main observations can be summarized as follows:

- (i) In 4 PD and 5 PD problems, the algorithm finds the optimum solution obtained by CPLEX in a shorter amount of time.
- (ii) In 6 PD problems, there are 3 instances for which CPLEX finds a better solution than the SA algorithm, but at an extensive amount of computational effort.
- (iii) For larger problems, CPLEX can not find any solution.

In summary, these tests have shown that the SA algorithm developed in this thesis can be used as a planning tool for real-life applications.

6. REAL-LIFE APPLICATION

Following the experimentation and having shown the good performance of the algorithm with empirical data, the SA algorithm is applied to Boğaziçi University (BU) Personnel Transportation System. As being one of the most reputable universities in Turkey, Boğaziçi University offers upper class academic education to more than 10,000 students in different campuses located in Istanbul. Due to the huge student population, there are many personnel working in those campuses. In order to transfer those personnel from their houses to each campus, a transportation network is established by a private transportation company.

BU has four campuses. They are South Campus, North Campus and Hisarüstü Campus which are in Bebek and Kandilli Campus which is in Çengelköy.

BU Personnel Transportation System Data obtained from the service company includes the number of vehicles that are transporting personnel from demand nodes to the main campus, the locations of the pickup points and the information of which personnel is transported by which vehicle. 1042 personnel are picked up from pickup nodes and transported to the campuses. 961 of them are Bebek personnel, the rest is Kandilli. There are 37 buses to serve. 31 of the buses are used for Bebek personnel transportation and the rest is for Kandilli. The capacities of vehicles are different, thus the fleet is heterogeneous.

In this study, South Campus, North Campus and Hisarüstü Campus are treated as one campus called Rumelihisarüstü Campus since they are very close to each others and the personnel transportation of this campus is under consideration.

The objective of this problem is to determine the optimal routes for transporting groups of personnel from specific pickup nodes to the delivery node, i.e. Rumelihisarüstü Campus.

Before solving the problem, the data is adjusted according to the heuristic code. (x,y) coordinates of the pickup nodes and delivery node, i.e. Rumelihisarüstü Campus are found at www.maps.google.com. Distances between any pair of nodes are calculated by 2-dimensional Euclidean distance function ($\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$) in the algorithm. The pickup nodes are the locations specified in the given data and there are 171 different pickup nodes. The transportation request of a pickup node is specified by the total number of personnel which are located at that node. Service times are assumed to be 1 sec for each personnel, so service time of a node is the calculation of service times of all personnel located at that node. It is assumed that pickup and delivery nodes have the same service time.

The adjusted data is illustrated in Table B.1 and Table B.2 in the Appendix.

After a solution has been found by the current algorithm, it has been observed that a vehicle could go to the main campus more than one in its route. In real-life problem, a vehicle goes to the main campus after picking up all its personnel. Since this is not applicable for the real-life service problem, some modifications have been done and a penalty have been added to the objective function for a vehicle not to go to a pickup location after the main campus.

The algorithm has been run for 60,000 iterations and it took 3734.452 seconds. The objective value is 21.904.

The solution of the problem is shown in Table 6.1. As it is seen in the solution, the number of used vehicles is less than the current number of vehicles in fleet. In other words, 27 of 31 are used to serve all personnel.

Table 6.1. The solution of BU personnel transportation system

NV	No of PD
31	134-48
2	165-4
5	61-5-42-82-101
7	62-156-55-69-151-46-158-128-34-159-7
11	75-171-3-45-96-36-108-121
13	56-137-153-26
17	127-116-66-92-39-8-110-17
19	54-76-35-122-43-70-10
9	71-129-21
10	139-9-117-102-123-104
21	143-2-130-11
25	22-133-152-146-78-31-125
27	112-23-113-64-19-67-60-118-38-131
29	114-68-119-32
18	30-126-117-89-47-51-138-132-59-103-91-164-147-85
4	37-16-15-14-141-136-88-142-53
3	72-170-107
30	140-100-120-90-94-57-12
26	18-50-79
23	154
8	44-6
15	162-40-83-168-144-80
14	73-87-58-20-81-160-27-49-98-97
28	157-77-33-25-24-74-52-124
20	28-116-111
22	105-145-63-86-150-148-163-1-135-29-106-84-149-93
6	169-65-95-99-155-161-115-109-41-13

7. CONCLUSIONS

7.1. Summary

In this study, VRPPDTW problems have been studied. Since it is NP-hard and it is not possible to find optimal solution for this type of problems, a heuristic approach has been developed and tested on firstly randomly generated problem instances and then applied for BU Personnel Transportation System.

SA algorithm has been used as solution procedure and two types of neighborhoods have been applied to search the solution space. Four different acceptance probability functions have been used and it has been observed that no one has given the best result for all problem instances.

The problem instances have been solved by CPLEX solver in GAMS and by developed algorithm. It has been observed that CPLEX could find a solution for only small size of problem instances whereas the developed algorithm have found optimal solutions for those problem instances within less CPU time and solved larger-size of problem instances. It is hard to say anything about the efficiency of the algorithm without solving benchmark problems in the literature and comparing the results.

Finally, the problem of BU Personnel Transportation System has been solved by the algorithm. The objective has been to determine the optimal routes for transporting groups of personnel from specific pickup nodes to the delivery node, i.e. Rumelihisariüstü Campus. At the result, an applicable solution which is different from the current application has been obtained.

7.2. Recommendations for Future Research

To see the effect of using a different heuristic algorithm, Tabu Search can be employed as a solution procedure for the same mathematical model. It is known that

Tabu Search is very effective heuristic in vehicle routing problems. To provide the convenience of the personnel, some constraints which are not considered in this study like route-length could be included.

APPENDIX A: RESULTS OF EXPERIMENTAL DESIGN

Table A.1. Routes found by CPLEX and the SA heuristic for 4 PD

	BHS		CPLEX SOLN.	
NPI	NV	Nodes	NV	Nodes
1	3	4-0-1-2-3-5	3	4-0-1-2-3-5
	4	6-7	4	6-7
2	2	0-1	2	0-1
	4	2-4-6-3-5-7	4	2-4-6-3-5-7
3	1	4-5	1	4-5
	2	0-2-3-1-6-7	2	0-2-3-1-6-7
4	1	4-5-0-1	1	4-5-0-1
	4	6-2-7-3	4	6-2-7-3
5	3	0-1	3	0-1
	4	4-6-2-7-3-5	4	4-6-2-7-3-5
6	3	0-1	3	0-1
	1	2-3	1	2-3
	4	6-7-4-5	4	6-7-4-5
7	1	2-0-1-3	1	2-0-1-3
	2	6-7	2	6-7
	3	4-5	3	4-5
8	1	6-4-0-5-7-1	1	6-4-0-5-7-1
	4	2-3	4	2-3
9	4	4-6-7-5	2	4-6-7-5
	1	0-2-3-1	3	0-2-3-1
10	2	2-6-0-7-1-3	1	4-5
	3	4-5	2	2-6-0-7-1-3

Table A.2. Routes found by CPLEX and the SA heuristic for 5 PD

	BHS		CPLEX SOLN	
NPI	NV	Nodes	NV	Nodes
1	1	2-3-6-7	1	2-3-6-7
	2	0-1	2	0-1
	5	8-4-5-9	5	8-4-5-9
2	1	8-9	1	8-9
	2	4-6-5-7	2	4-6-5-7
	4	2-0-1-3	4	2-0-1-3
3	2	4-2-5-3-8-9	2	4-2-5-3-8-9
	5	6-7-0-1	5	6-7-0-1
4	1	6-7-8-9	1	6-7-8-9
	3	2-3	3	2-3
	4	0-1	4	0-1
	5	4-5	5	4-5
5	2	8-0-2-9-3-4-5-1	2	8-0-2-9-3-4-5-1
	5	6-7	5	6-7
6	3	6-8-7-9	3	6-8-7-9
	4	4-0-2-1-3-5	4	4-0-2-1-3-5
7	5	2-8-0-3-9-4-6-5-1-7	5	2-8-0-3-9-4-6-5-1-7
8	3	8-4-0-5-2-3-9-1	3	8-4-0-5-2-3-9-1
	5	6-7	5	6-7
9	1	4-2-5-3	1	4-2-5-3
	2	8-6-7-9-0-1	2	8-6-7-9-0-1
10	1	4-5	1	4-5
	5	8-0-2-6-9-1-3-7	5	8-0-2-6-9-1-3-7

Table A.3. Routes found by CPLEX and the SA heuristic for 6 PD

	BHS		CPLEX SOLN	
NPI	NV	Nodes	NV	Nodes
1	1	6-4-7-5	1	6-4-7-5
	4	2-0-8-3-10-11-9-1	5	2-0-8-3-10-11-9-1
2	2	6-10-7-11	-	-
	1	0-2-4-1-3-5	-	-
	3	8-9	-	-
3	3	6-2-7-0-3	-	-
	2	8-4-5-9	-	-
	6	10-11	-	-
4	-	-	1	4-2-5-10-0-11-3-1
	-	-	4	8-6-7-9
5	2	10-11-8-2-9-4-3-5	2	10-11-8-2-9-4-3-5
	3	0-6-1-7	3	0-6-1-7
6	1	4-2-0-5-3-1	-	-
	5	8-10-11-9-6-7	-	-
7	6	0-4-5-1	-	-
	1	8-6-9-7	-	-
	4	2-3-10-11	-	-
8	-	-	1	4-10-5-11-8-2-3-6-9-7
	-	-	2	0-1
9	2	0-1-10-11	2	0-1-10-11
	4	4-8-5-2-9-6-7-3	4	4-8-5-2-9-6-7-3
10	-	-	1	2-3-8-4-5-9
	-	-	2	10-6-7-0-1-11

Table A.4. Routes found by the SA heuristic for 10 PD

	BHS	
NPI	NV	Nodes
1	5	0-2-3-6-7-1
	8	16-17-18-10-19-11
	9	14-15
	2	8-9-4-5-12-13
2	8	0-10-1-11
	2	4-5
	10	14-8-9-16-2-15-3-17
	6	6-18-19-12-7-13
3	3	12-13-2-3
	1	0-8-9-1
	5	18-10-19-11
	4	14-15
	8	16-17-4-5-6-7
4	6	18-19-10-16-17-11
	5	2-4-5-3
	3	8-6-7-9
	8	12-14-0-15-13-1
5	9	6-2-10-7-16-14-3-15-11-17
	3	18-8-0-9-4-5-19-1
	6	12-13

Table A.5. Routes found by the SA heuristic for 20 PD

	BHS	
NPI	NV	Nodes
1	17	30-28-4-29-32-33-38-14-34-15-39-35-5-31
	15	26-22-12-8-10-11-13-0-9-18-16-2-24-36-23-17-37-25-1-3-20-27-6-19-7-21
2	11	16-24-2-12-3-28-4-14-29-13-25-10-11-15-5-17
	18	22-30-6-38-39-23-31-7
	15	20-0-18-36-1-8-37-34-19-9-27-27-32-33-35-21
3	20	8-2-6-14-18-9-7-0-19-15-3-12-1-13
	2	20-32-28-21-10-22-23-38-39-36-26-37-29-33-11-27
	10	24-4-34-35-30-5-25-16-17-31
4	12	2-32-33-3-4-24-26-25-38-28-39-27-34-14-36-8-15-20-21-35-5-37-29-9
	7	10-30-11-31-12-0-22-23-1-16-17-18-13-6-7-19
5	7	2-10-18-30-3-19-31-11
	20	12-34-32-13-24-33-35-28-29-25
	18	20-16-14-4-15-26-21-6-5-27-17-7
	12	0-8-36-22-38-37-1-39-23-9

Table A.6. Routes found by the SA heuristic for 50 PD

	BHS	
NPI	NV	Nodes
1	16	76-26-27-77-40-58-60-84-14-41-62-42-22-86-87-85-82-2-18-63-613-83-43- 98-19-90-59-92-15-23-93-91-99-4-46-48-5-47-49
	35	36-12-96-50-78-79-32-97-24-6-7-80-81-56-37-33-13-94-51-25-27-10-0-44- 70-1-95-45-38-72-34-68-88-71-64-28-29-73-66-89-35-69-11-65-39-67
	32	8-74-16-54-75-9-30-20-52-53-55-17-21-31
2	1	8-60-82-24-22-34-90-83-2-10-28-6-80-25-9-23-7-3-81-29-94-26-61-36-98- 18-91-78-95-16-27-44-86-37-35-17-79-45-19-20-96-11-99-52-50-21-53-87 97-51
	47	76-38-14-72-15-54-4-39-77-84-55-46-48-88-73-5-8512-56-92-6847-62-64- 65-66-40-89-0-13-69-58-30-59-93-67-70-31-63-32-74-33-57-1-42-71-75-49- 43-41
	36	80-30-78-88-24-89-25-8-74-28-6-79-42-9-81-29-72-75-73-7-10-98-43-31-44- 45-11-99
3	43	94-40-84-52-90-53-68-91-18-69-48-66-85-41-38-39-95-0-64- 50-34-51-67-49-19-2-96-65-35-4-3-12-22-5-23-13-1-97-36-37
	10	56-58-92-93-32-14-15-70-71-60-82-59-57-83-76-62-54-33-20-77-61-21-63- 86-46-16-47-87-17-26-55-27
4	47	40-78-72-18-54-41-74-55-84-80-73-60-22-79-2-8-19-75-81-23-3-85-12-9-94 61-76-77-13-64-65-95
	10	44-34-16-45-26-27-96-90-68-35-4-5-17-38-30-91-46-31-62-88-63-47-42-43- 50-51-97-20-69-21-89-39
	49	66-67
	40	58-36-10-70-59-92-93-37-24-25-82-71-6-56-11-14-7-15-48-83-32-0-52-57 49-53-28-33-98-86-87-99-29-1
5	16	76-6-82-83-77-7-98-0-12-1-56-34-99-35-58-57-13-59
	37	78-14-66-15-96-44-97-67-32-8-45-79-48-9-54-49-62-33-60-10-18-11-19-63- 36-52-55-61-53-37
	38	50-24-51-25-84-85-64-40-70-88-68-89-71-90-91-65-69-41
	42	42-72-94-86-74-75-28-38-29-26-95-27-30-73-43-16-31-2-17-4-20-46-80-92- 93-47-39-5-22-3-81-87-23-21

Table A.7. Routes found by the SA heuristic for 75 PD

	BHS	
NPI	NV	Nodes
1	42	0-128-68-69-1-24-146-6-34-25-30-62-110-35-28-63-31-96-18-7-8- 46-111-60-100-9-52-53-47-70-61-71-29-140-141-19-147-94-101-42-95-66- 67-97-43-129
	69	56-126-80-81-54-138-50-127-57-78-120-44-139-92-93-51-121-82- 106-107-112-113-16-10-55-11-45-26-102-17-36-103-83-27-79-37
	56	76-86-130-134-58-59-22-135-104-118-87-122-105-4-132-64-123-119-5- 124-131-32-84-77-125-136-38-65-14-33-48-39-15-85-2-3-23-114-49-133- 115-137
	62	90-40-41-12-88-108-148-109-13-149-98-72-74-99-75-142-73-144-116-145- 91-89-117-20-21-143

Table A.8. Routes found by the SA heuristic for 100 PD

	BHS	
NPI	NV	Nodes
1	89	122-76-60-28-154-155-118-0-1-6-24-136-77-119-160-7-161-110-61-25- 116-111-112-117-172-113-70-137-4-173-128-123-29-129-48-71-5-49
	21	114-44-80-14-58-50-20-72-73-22-140-45-59-46-152- 51-141-132-133-81-21-56-23-15-82-83-115-153-166-57-167-47
	66	180-178-186-38-42-187-52-53-96-158-40-39-174-97-41-84-88- 159-10-62-175-34-179-43-11-35-148-85-149-89-63-181-64-65
	17	68-104-92-102-69-93-103-182-126-120-196-2-197-105-156- 157-124-108-194-127-195-125-54-55-170-192-30-31-183-121-171-3
	78	188-198-78-162-18-86-100-19-176-189-163-66-146-26-87- 27-147-130-134-135-184-94-177-95-16-17-101-185-79-67-131-199
	59	144-106-90-8-91-36-138-9-168-32-145-98-164-74-99-12-139-37-142-150- 190-13-75-151-33-107-191-143-165-169

APPENDIX B: DATA SET OF BU PROBLEM

Table B.1. Data set of BU personnel transportation system

No	PD	<i>pax.</i>	x	y	Service Time (sec)
1	63 Durağı	1	40.949509	29.10509	1
2	Acarlar	2	40.993326	29.15666	2
3	Acıbadem	10	41.001861	29.03871	10
4	Alibeyköy 1	43	41.079028	28.944254	43
5	Alibeyköy 2	35	41.079028	28.944254	35
6	Alibeyköy 3	34	41.079028	28.944254	34
7	Altıncöy	3	41.043029	29.07749	3
8	Altıntepe	3	40.953974	29.10439	3
9	Altunizade	8	41.019153	29.04747	8
10	Anadolu Hisarı	1	41.082117	29.06708	1
11	Arnavutköy	7	41.06806	29.041071	7
12	ATA 2	5	41.047609	29.07635	5
13	Atakent	1	41.015948	29.10716	1
14	Ataköy	11	40.983656	28.860226	11
15	Ataköy 5	2	40.982879	28.850784	2
16	Ataköy 9	1	40.985859	28.838768	1
17	Ataşehir	5	40.987932	29.13626	5
18	Avcılar	20	40.981453	28.72273	20
19	Ayşe Kadın	2	40.965868	29.08714	2
20	B.Çekmece	2	41.022683	28.57836	2
21	Bağcılar	10	41.045181	28.840313	10
22	Bağlar	2	41.026147	28.8204	2
23	Bağlarbaşı	3	41.064469	28.91499	3
24	Bahçelievler	14	41.004516	28.867264	14
25	Bakırköy	1	40.98074	28.834648	1
26	BaltalimanıHastanesi	5	41.097464	29.053345	5

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
27	Başak Konutları	10	41.092548	28.979874	10
28	Bayrampaşa	3	41.051654	28.900566	3
29	Beşiktaş	2	41.05133	29.008369	2
30	Beşyol	1	40.990686	28.79718	1
31	Beykoz	7	41.126177	29.10072	7
32	Beylerbeyi	1	41.044671	24.0496	1
33	Beylikdüzü	3	40.982036	28.63986	3
34	Birlik Mah.	9	41.027215	29.07397	9
35	Bostancı	11	40.957855	29.09568	11
36	Bulgurlu	2	41.014944	29.07227	2
37	Büyükkçekmece	1	41.022683	28.57836	1
38	Büyükdere	4	41.163859	29.042315	4
39	Caddebostan	3	40.96684	29.06279	3
40	Çağlayan	1	41.077604	28.982449	1
41	Çakmak	5	41.014005	29.11965	5
42	Çamlıca	1	41.022456	29.06377	1
43	Çamlık	4	41.017599	29.19641	4
44	Çapa	2	41.014572	28.935177	2
45	Capitol	1	41.020562	29.03944	1
46	Çekmeköy	3	41.032945	29.17377	3
47	Çeliktepe	3	41.084203	29.000301	3
48	Çengelköy	4	41.049275	29.05429	4
49	Çiçekçi	2	41.0136	29.01485	2
50	Çiftehavuzlar	4	40.970348	29.05303	4
51	Dikilitaş	2	41.056055	29.00425	2
52	Doğancılar	2	41.022302	29.01066	2
53	Dudullu	1	41.017632	29.1752	1

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
54	Edirnekapı	3	41.031489	28.936057	3
55	Erenköy	7	40.972892	29.07534	7
56	Esenkent	7	40.929467	29.1663	7
57	Esenyalı	1	40.860134	29.29302	1
58	Esenyurt	3	41.035114	28.67028	3
59	Ethemefendi	1	40.976253	29.0759	1
60	Etiler	2	41.084268	29.035149	2
61	Eyüp	4	41.040366	28.92742	4
62	Fatih	5	41.019769	28.947537	5
63	Fenerbahçe	1	40.974512	29.04341	1
64	Feneryolu	2	40.982134	29.04953	2
65	Fikirtepe	3	40.993747	29.05036	3
66	Fındıkzade	5	41.01292	28.94024	5
67	Fıstıkağacı	5	41.028284	29.02764	5
68	Gazi Osman Paşa	5	41.028284	29.02764	5
69	Gebze	3	40.799257	29.4389	3
70	Göksu Evleri	1	41.080969	29.08759	1
71	Göztepe	18	40.977047	29.062614	18
72	Gültepe	11	41.079513	28.993435	11
73	Güngören	3	41.019113	28.883443	3
74	Halıcıoğlu	3	41.048709	28.94898	3
75	Halkalı	12	41.025629	28.790617	12
76	Hasippaşa	2	41.038675	29.07334	2
77	Haznedar	4	41.010508	28.868508	4
78	İçerenköy	7	40.970761	29.11111	7
79	İdealtepe	4	40.940457	29.12004	4
80	Ihlamurkuyu	1	41.024366	29.13639	1

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
81	İkitelli	6	41.075825	28.787141	6
82	İmar İskan Bl.	1	40.992208	29.14546	1
83	İnönü Cad.	5	40.979639	29.08781	5
84	İstinye Bayırı	7	41.11124	29.036264	7
85	İstinye Koç Üniversitesi	3	41.12081	29.049225	3
86	K.Bakkalköy	1	40.992403	29.11489	1
87	K.Karabekir	1	41.02796	28.843918	1
88	Kadıköy	15	40.981194	29.08785	15
89	Kağıthane	5	41.083427	28.978329	5
90	Kalamış	1	40.978732	29.04144	1
91	Kanlıca	1	41.099437	29.07279	1
92	Kartal	9	40.911177	29.15943	9
93	Kavacık	5	41.089638	29.08467	5
94	Kaynarca	7	40.868029	29.2614	7
95	Kazasker	3	40.969578	29.08444	3
96	Kirazlıtepe	6	41.03976	29.06339	6
97	Kızıltoprak	2	40.983373	29.03967	2
98	Koşuyolu	2	41.006654	29.03459	2
99	Kozyatağı	1	40.968946	29.09566	1
100	Küçükköy	18	41.074806	28.897219	18
101	Küçüksu	4	41.068869	29.07201	4
102	Küçükyalı	14	40.949615	29.10976	14
103	Küplüce	1	41.036603	29.05489	1
104	Kurtköy	9	40.91494	29.29792	9
105	Kurtuluş	7	41.049906	28.982062	7
106	Kuruçeşme	4	41.060942	29.03532	4
107	Kuyubaşı	8	40.984304	29.05047	8

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
108	Libadiye	5	40.993893	290736	5
109	Madenler	20	41.014086	29.18973	20
110	Maltepe	8	40.950603	29.15531	8
111	Mecidiyeköy	28	41.07191	28.996696	28
112	Meskenler	1	41.016952	28.89434	1
113	Moda	5	40.984806	29.02455	5
114	Mustafapaşa	14	41.004419	28.933225	14
115	Nato Yolu	2	40.998735	29.18179	2
116	Nurtepe	3	41.074531	28.959961	3
117	Örnek Mah.	2	40.998201	29.08012	2
118	Ortaçeşme	3	41.145214	29.09317	3
119	Ortaköy	16	41.054049	29.028411	16
120	Osmanbey	4	41.049761	28.988392	4
121	Paşabahçe	3	41.114764	29.09218	3
122	Pendik	14	40.879256	29.25591	14
123	Rahmanlar	3	40.898479	29.17776	3
124	Rasathane	8	41.063336	29.06474	8
125	Reşitpaşa	7	41.105484	29.035664	7
126	S.Çiftliği	6	41.09727	28.85937	6
127	Sanayi Mahallesi	4	41.021485	28.880181	4
128	Santral	3	41.024755	29.08867	3
129	Saraçhane	3	41.014669	28.954403	3
130	Sarıgazi	9	41.007204	29.214749	9
131	Sarıyer	16	41.17271	29.024162	16
132	Şaşkınbakkal	1	40.963195	29.07536	1
133	Sefaköy	7	41.001148	28.79697	7
134	Şehremini	5	41.015883	28.932109	5

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
135	Selimiye	2	41.006249	29.01738	2
136	Şenesenevler	4	40.960853	29.0933	4
137	Şerifali	10	41.011382	29.15274	10
138	Setbaşı	1	41.027142	29.02474	1
139	Seyrantepe	6	41.095742	28.992501	6
140	Şirinevler	3	40.999108	28.846965	3
141	Şişli	6	41.058774	28.987899	6
142	Soyak	1	41.009277	29.11476	1
143	Suadiye	1	40.961072	29.0815	1
144	Sultanbeyli	18	40.964475	29.2693	18
145	Taksim	4	41.037033	28.985077	4
146	Talimhane	3	41.068497	29.06588	3
147	Tarabya	4	41.135098	29.046478	4
148	Taşdelen	3	41.031845	29.22466	3
149	Tokatköy	2	41.161985	29.10107	2
150	Türk İş Bl.	2	40.988418	29.15347	2
151	Tuzla	3	40.891196	29.35547	3
152	Ulus	3	41.064533	29.026694	3
153	Ümraniye	17	41.030291	29.10536	17
154	Üsküdar	16	41.034823	29.04897	16
155	Üst.Bostancı	1	40.960456	29.10218	1
156	Validebağ	1	41.014912	29.04682	1
157	Vatan Cad.	3	41.020189	28.93593	3
158	Y.Dudullu	3	41.00782	29.14681	3
159	Yavuztürk	3	41.04277	29.08498	3
160	Yayla	10	41.142984	28.862457	10
161	Yeni Mah.	3	40.922714	29.12897	3

Table B.1 – (cont.)

No	PD	<i>pax.</i>	x	y	Service Time (sec)
162	Yenibosna	6	41.010346	28.8307	6
163	Yenidoğan	1	41.012936	29.24131	1
164	Yeniköy	2	41.122556	29.058838	2
165	Yeşilkent	1	41.038999	28.70015	1
166	Yeşilova	1	40.992532	28.7871	1
167	Yeşilpınar	13	41.074628	28.920565	13
168	Yetmişevler	2	40.93078	29.14495	2
169	Yıldızbakkal	4	40.994978	29.03324	4
170	Z.Kamil	4	41.016514	29.02585	4
171	Ziverbey	4	40.988767	29.04287	4

Table B.2. The capacities of buses in BU personnel transportation system

NV	VC	NV	VC
1	29	17	45
2	45	18	45
3	45	19	45
4	45	20	45
5	45	21	18
6	45	22	29
7	45	23	45
8	45	24	27
9	45	25	45
10	45	26	45
11	45	27	45
12	27	28	45
13	45	29	45
14	45	30	45
15	45	31	45
16	45		

APPENDIX C: THE CODE OF SA ALGORITHM

```

#include <iostream>
#include <stdio.h>
#include <vector>
#include "time.h"
#include "math.h"
using namespace std;

#define T_reduction 0.9
#define max_iteration_for_SA 100
#define treshold_T 250
#define max_number_of_temperature_reduction 10
#define request_size 171
#define vehicle_size 171

int a=0;
int b=0;
int c=0;
int d=0;
int e=0;
int f=0;

vector<double>data_capacity;
vector<double>data_speed;
vector<double>data_load;
vector<double>data_servicetime;
vector<double>data_source_xy;
vector<double>data_destination_xy;

class vehicle;
class solution;
class problem;
class point;
class route;
class request;
class manager;

```

```
class algorithm;
```

```
class manager
```

```
{
    problem* myproblem;
    solution* mysolution;
public:
    void run();
    problem* createinstance(/*int seed,*/ int number_of_vehicles,int number_of_requests);
};
```

```
class problem
```

```
{
    vector<point*> mypoints;
    vector<request*> myrequests;
    vector<vehicle*> myvehicles;
public:
    problem(int v, int r);
    vehicle* returnvehicle(int a);
    point* returnsource(int a);
    point* returndestination(int a);
    point* returnpoint(int);
    int readvehiclesize();
    int readpointssize();
    int readrequestsize();
    void writemypoints();
    void writetofile();
    request* returnrequest(int a);
};
```

```
class solution
```

```
{
    double total_fx;
public:
    vector<route*> myroutes;
    problem* myproblem;
    void assignsolution(solution*);
```

```

void assignsolution(solution);
solution();
solution(problem* aproblem);
solution(const solution&);
vector<solution*> solve();
void writetofile();
void writetxtotofile();
void calculate_total_fx();
double returntotalfx();
int return_request_of_route_id(request);
void createinitialroutes();
void initialize();
void exchange(request,request);
void exchange_for_same_destination(request,request);
void exchange2(request,request);
void exchange2_for_same_destination(request,request);
bool merge(request,request);
void merge2(request,request);
void merge3(request,request);
bool is_feasible();
bool is_requestfeasible();
bool is_routefeasible();
bool is_requestssameroute(request,request);
void print_total_fx();
solution* variatesolution1();
vector<solution*> variatesolution();
solution* variatesolution2();
route* returnroute(int);
int returnroutenumber();
vector<route*>::iterator findrouteit(route*);
vector<route*>returnmyroutes();
double find_initial_T();
bool is_vehicle_assigned(vector<vehicle*> vehiclelist, vehicle*);
void write();
void setproblem(problem*);
};
class point

```

```

{
    double x;
    double y;
    double load;
    double earliestarrival;
    double latestarrival;
    double servicetime;
    bool is_assigned;
    bool is_depot;
    request* myrequest;
public:
    void setdatamembers(double lb, double ub, double l, double st);
    void setx(double);
    void sety(double);
    void set_is_depot(int);
    void random_generation();
    point(double a, double b);
    point();
    double whatisyourload();
    double returnx();
    double returny();
    double returnservicetime();
    double returnearliestarrival();
    double returnlatestarrival();
    void definerequest(request* owner);
};

```

```

class route
{
    vehicle* myvehicle;
    double fx;
    problem* myproblem;
public:
    vector<point*> mypoints;
    void writerroute();
    void assignroute(route* newroute, route* aroute);
    void assign_vehicle_from_myvehicles(vehicle *);

```

```

void assign_vehicle_from_myvehicless(vehicle*);
void addpointtoroute(point*);
void calculate_fx();
void change_points(int,point*);
vehicle* returnvehicle();
point* returnpoint(int);
double returnfx();
double calculate_distance(point*,point*);
bool is_timefeasible();
bool is_loadfeasible();
int returnsizeofpoints();
bool is_feasible();
bool is_orderfeasible();
void add(vector<point*>::iterator, point* apoint);
route add(request a);
route remove(request b);
vector<point*> eraserequest(point* apoint);
vector<point*> returnmypoints();
void changemypoints(vector<point*>);
vector<point*>::iterator returnbegin();
vector<point*>::iterator returnend();
route(problem*);
int returnpointindeks(point*);
};

```

```

class request
{
    point* source;
    point* destination;
    double load;
    double earliestpickup;
    double latestpickup;
    double earliestdelivery;
    double latestdelivery;
    double servicetime;
public:

```

```

        request();
        point* readsource();
        point* readdestination();
    };

class vehicle
{
    double capacity;
    double speed;
    point startingpoint;
    point endingpoint;
    route* myroute;
public:
    vehicle();
    point returnstartingpoint();
    point returnendingpoint();
    double returncapacity();
    double returnspeed();
};

class algorithm
{
public:
    solution* solvewithSA2(vector<solution*> );
    void solve(solution*);
};

void route::assign_vehicle_from_myvehicles(vehicle* vehiclefrommyvehicles)
{ myvehicle=vehiclefrommyvehicles; }

void route:: writeroute()
{
    FILE* cfPtr;
    if((cfPtr=fopen("data.txt","a"))==NULL)
        cout<<"The file can not be opened"<<endl;
    else
    {

```

```

        for(int i=0;i<returnsizeofpoints();i++)
            fprintf(cfPtr,"%d\n",returnpoint(i)->returnx());

        fprintf(cfPtr,"%d\n",returnvehicle()->returncapacity());
    }
    fclose(cfPtr);
}

void route::assign_vehicle_from_myvehicless(vehicle* vehiclefrommyvehicles)
{ *myvehicle=*vehiclefrommyvehicles; }

void route:: addpointtoroute(point* p)
{ mypoints.push_back(p); }

void route::add(vector<point*>::iterator iter1, point* apoint)
{ mypoints.insert(iter1,apoint); }

vector<point*>::iterator route:: returnbegin()
{ return mypoints.begin(); }

vector<point*>::iterator route::returnend()
{ return mypoints.end(); }

vehicle* route::returnvehicle()
{ return myvehicle; }

point* route::returnpoint(int a)
{ return mypoints[a]; }

int route::returnsizeofpoints()
{ return mypoints.size(); }

void route::change_points(int index,point* changed_point)
{ mypoints[index]=changed_point; }

```

```

vector<point*> route::eraserequest(point* apoint)
{
    vector<point*>::iterator iter,iter2;
    int cont=1;
    int i=0;
    vector<point*> copy_mypoints(mypoints);
    for(iter=mypoints.begin();cont==1;)
    {
        iter2=copy_mypoints.begin()+i;
        if(*iter2==apoint)
        {
            copy_mypoints.erase(iter2);
            cont=0;
        }
        i++;
        if(iter==mypoints.end()) cont=0;
        if(cont==1) iter++;
    }
    return copy_mypoints;
}

route::route(problem* aproblem)
{ myproblem=aproblem; }

int route::returnpointindeks(point* apoint)
{
    for(int i=0;i<this->myproblem->readpointssize();i++)
        if (apoint==this->myproblem->returnpoint(i))
            int a=i;
    return a;
}

//The modification for the service problem

vector<point*> convert(vector<point*> mypoints)
{

```

```

vector<point*> temp_mypoints;
for(int i=0;i<mypoints.size();i++)
{
    if(mypoints[i]->whatisyourload()>0) temp_mypoints.push_back(mypoints[i]);
}
int cont=1;
for(int i=0;cont==1;i++)
{
    if(mypoints[i]->whatisyourload()<0)
    {
        temp_mypoints.push_back(mypoints[i]);
        cont=0;
    }
}
return temp_mypoints;
}

```

//The modification for the service problem

```

void convert(vector<route*> myroutes)
{
    for(int i=0;i<myroutes.size();i++)
    {
        vector<point*> temp=convert(myroutes[i]->returnmypoints());
        myroutes[i]->mypoints=temp;
    }
}

```

//The modification for the service problem

```

void route::calculate_fx()
{
    vector<point*> temp_mypoints(convert(mypoints));
    //a temporary vector which has initial and destination depot
    point vehicle_starting_point=myvehicle->returnstartingpoint();
    point vehicle_ending_point=myvehicle->returnendingpoint();
    temp_mypoints.insert(temp_mypoints.begin(),&vehicle_starting_point);
}

```

```

temp_mypoints.insert(temp_mypoints.end(), &vehicle_ending_point);

fx=0;

for(int a=0;a<(temp_mypoints.size()-1);a++)
    fx=fx+calculate_distance(temp_mypoints[a],temp_mypoints[a+1]);
fx=fx/(myvehicle->returnspeed());

for(int i=0;i<temp_mypoints.size();i++)
    fx=fx+temp_mypoints[i]->returnservicetime();
}

double route::returnfx()
{
    calculate_fx();
    return fx;
}

vector<point*> route::returnmypoints()
{ return mypoints; }

void route::changemypoints(vector<point*> apointlist)
{ mypoints=apointlist; }

double route::calculate_distance(point* first, point* second)
{
    return sqrt(pow( ( (first-> returnx() ) - (second-> returnx() ) ),2)+pow( ( (first-> returny() )-(second->
returny() )),2));
}

bool route::is_loadfeasible()
{
    bool result=1;

    // a temporary vector which has initial and destination depot
    vector<point*> temp_mypoints(mypoints);
    point vehicle_starting_point=myvehicle->returnstartingpoint();

```

```

    point vehicle_ending_point=myvehicle->returnendingpoint();
    temp_mypoints.insert(temp_mypoints.begin(),&vehicle_starting_point);
    temp_mypoints.insert(temp_mypoints.end(), &vehicle_ending_point);
    double route_of_load=0;
    for(int i=0;i<temp_mypoints.size()&& result==1;i++){
        route_of_load=route_of_load+temp_mypoints[i]->whatisyourload();
        if(route_of_load>myvehicle->returncapacity())
            result=0;
    }

    return result;
}

//The modification for the service problem
bool route::is_timefeasible()
{
    bool result=1;
    //a temporary vector which has initial and destination depot
    vector<point*> temp_mypoints(convert(mypoints));
    point vehicle_starting_point=myvehicle->returnstartingpoint();
    point vehicle_ending_point=myvehicle->returnendingpoint();
    temp_mypoints.insert(temp_mypoints.begin(),&vehicle_starting_point);
    temp_mypoints.insert(temp_mypoints.end(), &vehicle_ending_point);
    double arrival_time=0;
    for(int i=0;i<(temp_mypoints.size()-1)&& result==1;i++){
        arrival_time=arrival_time+calculate_distance(temp_mypoints[i],temp_mypoints[i+1])/(myvehicle->
returnspeed())+2*(temp_mypoints[i]->returnservicetime());
        arrival_time=arrival_time+temp_mypoints[temp_mypoints.size()-1]->returnservicetime();
        if(arrival_time>temp_mypoints[i+1]->returnlatestarrival())
            result=0;
    }
    return result;
}

/*
bool route::is_timefeasible()
{
    bool result=1;

```

```

//a temporary vector which has initial and destination depot
vector<point*> temp_mypoints(mypoints);
point vehicle_starting_point=myvehicle->returnstartingpoint();
point vehicle_ending_point=myvehicle->returnendingpoint();
temp_mypoints.insert(temp_mypoints.begin(),&vehicle_starting_point);
temp_mypoints.insert(temp_mypoints.end(), &vehicle_ending_point);
double arrival_time=0;
for(int i=0;i<(temp_mypoints.size()-1)&& result==1;i++){
    arrival_time=arrival_time+calculate_distance(temp_mypoints[i],temp_mypoints[i+1])/(myvehicle->
returnspeed())+temp_mypoints[i]->returnservicetime();
        arrival_time=arrival_time+temp_mypoints[temp_mypoints.size()-1]->returnservicetime();
        if(arrival_time>temp_mypoints[i+1]->returnlatestarrival())
            result=0;
    }
    return result;
}
*/
bool route::is_feasible()
{
    bool result;
    vector<route*> temp;
    temp.push_back(this);
    convert(temp);
    result = is_orderfeasible()*is_timefeasible()*is_loadfeasible();
    return result;
}

bool route::is_orderfeasible()
{
    bool result=1;

    for(int i=0;i<mypoints.size() && result==1 ;i++)
    {
        if(mypoints[i]->whatisyourload())<0)
        {
            result=0;
            for(int j=0;j<i;j++)

```

```

        if(-(mypoints[i]->whatisyourload())==mypoints[j]->whatisyourload())
            result=1;
    }
}
return result;
}

void route::assignroute(route* newroute, route* aroute)
{
    newroute->mypoints=aroute->mypoints;
    newroute->myvehicle=aroute->myvehicle;
}

vehicle::vehicle()
{
    capacity=0;
    /*while(capacity<100)
        capacity=(double)rand()/RAND_MAX*200; */
    capacity=data_capacity[a];
    a=a+1;

    //speed=0;
    //while(speed<1)
        //speed=(double)rand()/RAND_MAX*2;
    //speed=data_speed[b];
    //b=b+1;
    speed=1;
    startingpoint.setdatamembers(0,50,0,0);
    endingpoint.setdatamembers(0,50,0,0);
    startingpoint.setx(41.020);
    startingpoint.sety(28.918);
    endingpoint.setx(41.020);
    endingpoint.sety(28.918);
    myroute=NULL;
}

point vehicle::returnstartingpoint()

```

```

{ return startingpoint; }
point vehicle::returnendingpoint()
{ return endingpoint; }

double vehicle::returnspeed()
{ return speed; }

double vehicle::returncapacity()
{ return capacity; }
problem::problem(int v, int r)
{
    for(;myvehicles.size()!=v;myvehicles.push_back(new vehicle)){ }
    for(;myrequests.size()!=r;myrequests.push_back(new request)){ }
    for(int a=0;a<myrequests.size();a++){
        mypoints.push_back(myrequests[a]->readsource());
        mypoints.push_back(myrequests[a]->readdestination());
    }
    //The points and vehicles are written into .txt file.
    writetofile();
}

int problem::readvehiclesize()
{ return myvehicles.size(); }

int problem::readpointssize()
{ return mypoints.size(); }

int problem::readrequestsize()
{ return myrequests.size(); }

vehicle* problem::returnvehicle(int a)
{ return myvehicles[a]; }

point* problem::returnsource(int a)
{ return myrequests[a]->readsource(); }

request* problem::returnrequest(int a)

```

```

{ return myrequests[a]; }

point* problem::returnpoint(int a)
{ return mypoints[a]; }

point* problem::returndestination(int a)
{ return myrequests[a]->readdestination(); }

void problem:: writetofile()
{
    FILE* cfPtr;
    if((cfPtr=fopen("data.txt","w"))==NULL)
        cout<<"The file can not be opened"<<endl;
    else

        //written of points

        fprintf(cfPtr,"%25s%20s%20s%20s%20s%20s","x","y","load","service_time","earliest_arrival","latest_
arrival");

        for(int a=0;a<mypoints.size();a++){
            fprintf(cfPtr,"\n");
            if(a<10)
                fprintf(cfPtr,"%s%d%16.3f%20.3f%20.3f%20.3f%20.3f","Pointer
",a,mypoints[a]->returnx(),mypoints[a]->returny(),mypoints[a]->whatisyourload(),mypoints[a]->
returnservicetime(), mypoints[a]-> returnearliestarrival(),mypoints[a]-> returnlatestarrival());
            else
                fprintf(cfPtr,"%s%d%15.3f%20.3f%20.3f%20.3f%20.3f","Pointer
",a,mypoints[a]->returnx(),mypoints[a]->returny(),mypoints[a]->whatisyourload(),mypoints[a]-
>returnservicetime(),mypoints[a]->returnearliestarrival(),mypoints[a]->returnlatestarrival());

        }

        //written of vehicles
        fprintf(cfPtr,"\n\n\n\n");
        fprintf(cfPtr,"%25s%10s%85s%85s\n","capacity","speed","starting_depot
", "destination_depot");

```

```

        fprintf(cfPtr, "%4s%10s%10s%15s%20s%20s%10s%10s%10s%15s%20s%20s", "x", "y", "load", "service
_time", "earliest_arrival", "latest_arrival", "x", "y", "load", "service_time", "earliest_arrival", "latest_arrival");
        fprintf(cfPtr, "\n");
        for(int a=0; a<myvehicles.size(); a++){
            fprintf(cfPtr, "\n");
            if(a<10)

                fprintf(cfPtr, "%s%d%16.3f%10.3f%10.3f%10.3f%10.3f%15.3f%20.3f%20.3f%10.3f%10.3f%10.3f%15
.3f%20.3f%20.3f", "Vehicle", a, myvehicles[a]->returncapacity(), myvehicles[a]->returnspeed(), (myvehicles[a]-
>returnstartingpoint()).returnx(), (myvehicles[a]->returnstartingpoint()).returny(), (myvehicles[a]-
returnstartingpoint()).whatisyourload(), (myvehicles[a]->returnstartingpoint()).returnservicetime(), (myvehicles[a]-
>
                returnstartingpoint()).returnearliestarrival(), (myvehicles[a]->
returnstartingpoint()).returnlatestarrival(), (myvehicles[a]->
returnendingpoint()).returnx(), (myvehicles[a]->
returnendingpoint()).returny(), (myvehicles[a]->
returnendingpoint()).whatisyourload(), (myvehicles[a]->
returnendingpoint()).returnservicetime(), (myvehicles[a]->
returnendingpoint()).returnearliestarrival(),
(myvehicles[a]->returnendingpoint()).returnlatestarrival());
            else
                fprintf(cfPtr, "%s%d%15.3f%10.3f%10.3f%10.3f%10.3f%15.3f%20.3f%20.3f%10.3f%10.3f%10.3f%15
.3f%20.3f%20.3f", "Vehicle", a, myvehicles[a]->returncapacity(), myvehicles[a]->returnspeed(), (myvehicles[a]-
>returnstartingpoint()).returnx(),
                (myvehicles[a]->
returnstartingpoint()).returny(), (myvehicles[a]->
returnstartingpoint()).whatisyourload(), (myvehicles[a]->returnstartingpoint()).returnservicetime(), (myvehicles[a]-
>
returnstartingpoint()).returnearliestarrival(), (myvehicles[a]->
returnstartingpoint()).
returnlatestarrival(),
(myvehicles[a]->
returnendingpoint()).returnx(), (myvehicles[a]->
returnendingpoint()).returny(), (myvehicles[a]-
>
returnendingpoint()).whatisyourload(), (myvehicles[a]->
returnendingpoint()).returnservicetime(),
(myvehicles[a]
->
returnendingpoint()).returnearliestarrival(),
(myvehicles[a]->
returnendingpoint()).
returnlatestarrival());
        }
        fprintf(cfPtr, "\n");
        fclose(cfPtr);
    }
    void solution:: writexftofile()
    {
        FILE* cfPtr;
        if((cfPtr=fopen("data.txt", "a"))==NULL)
            cout<<"The file can not be opened"<<endl;
        else

```

```

        fprintf(cfPtr, "\n");
        fprintf(cfPtr, "%s%5.3f", "Fx of the solution: ", returntotalfx());

        fclose(cfPtr);
    }
    solution::solution()
    { }
    solution::solution(problem* aproblem)
    { myproblem=aproblem; }

    void solution::setproblem(problem* aproblem)
    { myproblem=aproblem; }

    solution::solution(const solution & asolution)
    {
        myproblem=asolution.myproblem;
        for(int i=0; i<myroutes.size(); i++)
            myroutes[i]=asolution.myroutes[i];
    }

    void solution::calculate_total_fx()
    {
        total_fx=0;
        for(int a=0; a<myroutes.size(); a++)
            total_fx=total_fx+myroutes[a]->returnfx();
    }
    double solution::returntotalfx()
    {
        calculate_total_fx();
        return total_fx;
    }

    int solution::return_request_of_route_id(request arequest)
    {
        int request_route_id;
        for(int i=0; i<myroutes.size(); i++)
            for(int j=0; j<myroutes[i]->returnsizeofpoints(); j++)

```

```

        if((myroutes[i]->returnpoint(j))==arequest.readsource())
            request_route_id=i;
    return request_route_id;
}

route* solution::returnroute(int route_id)
{ return myroutes[route_id]; }

vector<route*> solution::returnmyroutes()
{ return myroutes; }

vector<route*>::iterator solution:: findrouteit(route* aroute)
{
    vector<route*>::iterator it;
    for(it=myroutes.begin();it!=myroutes.end();it++)
        if(*it==aroute)
            return it;
}

double solution::find_initial_T()
{
    vector<route*> routelist;
    for(int i=0;i<50;i++)
    {
        int x= rand()%(myproblem->readvehiclesize());
        route* aroute=new route(myproblem);
        aroute->assign_vehicle_from_myvehicles(myproblem->returnvehicle(x));
        vector<int> list;
        for(int i=0;i<myproblem->readrequestsize();i++)
            list.push_back(i);
        for (int x=myproblem->readrequestsize();x>1;x--) //x is the number of element in the array
        {
            int rand_ind=rand()%x;
            int temp=list[rand_ind];
            list[rand_ind]=list[x-1];
            list[x-1]=temp;
        }
    }
}

```

```

        int k=0;

        for(int it=0;it<myproblem->readrequestsize();it++)
        {
            aroute->addpointtoroute(myproblem->returnrequest(list[k])->readsource());
            aroute->addpointtoroute(myproblem->returnrequest(list[k])->readdestination());
            k++;
        }
        routelist.push_back(aroute);
    }
    double average=0;
    for(int i=0;i<routelist.size();i++)
    {
        average=average+routelist[i]->returnfx();
    }
    routelist.clear();
    average=average/50;
    return average;
}

```

```

int solution::returnroutenumber()
{ return myroutes.size(); }

```

```

void algorithm::solve(solution* mysolution)
{ solvewithSA2(mysolution->solve()); }

```

```

solution* algorithm::solvewithSA2(vector<solution*> solutionpool)
{
    int number_of_iterations=0;
    int number_of_temperature_iterations=0;
    double initial_T=solutionpool[0]->find_initial_T();
    double T=initial_T;
    double alpha=T_reduction;
    int max_iteration_for_temperature=20;

    solution current;

```

```

current.setproblem(solutionpool[0]->myproblem);

solution copy_current;
copy_current.setproblem(solutionpool[0]->myproblem);

solution* best=new solution(solutionpool[0]->myproblem);

best->myroutes=solutionpool[0]->myroutes;

solution next;
copy_current.setproblem(solutionpool[0]->myproblem);
for(int i=0;i<solutionpool.size();i++)
{
    T=initial_T;
    number_of_ iterations=0;
    number_of_ temperature_ iterations=0;
    max_ iteration_ for_ temperature=100;

    best->writetofile();
    best->writefxtofile();

    current=*solutionpool[i];
    copy_current=*solutionpool[i];

    if(current.returntotalfx()<best->returntotalfx())
        current.assignsolution(best);

    for(; number_of_ iterations<=max_ iteration_ for_ SA ||T>threshol T ;)
    {
        for( ;number_of_ temperature_ iterations<= max_ iteration_ for_ temperature;)
        {
            next=(copy_current.variatesolution2());
            if(current.returntotalfx()>=next.returntotalfx())
            {
                current=next;
                copy_current=next;
            }
        }
    }
}

```

```

        else
        {
            //acceptance probability
            double acceptance_probability=exp(-1*(next.returntotalfx()-
current.returntotalfx()))/(T*current.returntotalfx());
            //1-((next->returntotalfx()-current->returntotalfx())/T);
            //exp(-1*(next->returntotalfx()-current->returntotalfx())/T);
            double x=(double)rand()/RAND_MAX;
            if(x<acceptance_probability)
            { copy_current=next; }
            else
            { copy_current=current; }
        }
        if(current.returntotalfx()<=best->returntotalfx())
        { current.assignsolution(best); }
        number_of_iterations++;
        number_of_temperature_iterations++;
    }
    number_of_temperature_iterations=0;

    T=T/(1+0.1*sqrt(T));

    //T=T*T_reduction;
    max_iteration_for_temperature=(int)(max_iteration_for_temperature*1.1);
}
}
return best;
}

bool solution::is_vehicle_assigned(vector<vehicle*> vehiclelist, vehicle* avehicle)
{
    bool result=1;
    for(int i=0;i<vehiclelist.size()&& result==1;i++)
        if(avehicle==vehiclelist[i])
            result=0;
    return result;
}

```

```

void solution::initialize()
{
    createinitialroutes();
    vector<solution*> solutionpool;
    int best_index=0;
    solution* temp;
    for(int a=0;a<2;)
    {
        temp=new solution(myproblem);
        temp=variatesolution1();
        if(temp->is_feasible())
        {
            solutionpool.push_back(temp);
            a++;
        }
    }
    for(int b=0;b<solutionpool.size();b++)
    {
        if(solutionpool[b]->returntotalfx()< solutionpool[best_index]->returntotalfx())
            best_index=b;
    }
    myroutes=solutionpool[best_index]->myroutes;

    FILE* cfPtr;
    if((cfPtr=fopen("data.txt","a"))==NULL)
        cout<<"The file can not be opened"<<endl;
    else{
        fprintf(cfPtr,"\n\n");
        fprintf(cfPtr,"%50s","BEST INITIAL SOLUTION");
    }
    fclose(cfPtr);
    writetofile();
    writefxtofile();
}

solution* solution::variatesolution1()
{

```

```

solution* newsolution=new solution(myproblem);
assignsolution(newsolution);

for(int non_improvement=0;non_improvement<2 ;)
{
    start:
    int rand1= rand()%(newsolution->myproblem->readrequestsize());
    int rand2= rand()%(newsolution->myproblem->readrequestsize());

    if(rand1==rand2)
        goto start;

    request request1= *(newsolution->myproblem->returnrequest(rand1));
    request request2=*(newsolution->myproblem->returnrequest(rand2));

    if(!newsolution->is_requestssameroute(request1,request2))
    {
        if(newsolution->merge(request1,request2)==1) non_improvement=0;
        else
        { non_improvement++; }
    }
}
return newsolution;
}

vector<solution*> solution::variatesolution()
{
    solution* newsolution=new solution(myproblem);
    assignsolution(newsolution);

    solution* copy_newsolution=new solution(myproblem);
    assignsolution(copy_newsolution);

    vector<solution*>solutionpool;

    int k=request_size;

```

```

for( int i=0;i<30;i++)
{
    start:

    int rand1= rand()%(newsolution->myproblem->readrequestsize());
    int rand2= rand()%(newsolution->myproblem->readrequestsize());

    if(rand1==rand2)
        goto start;

    request request1= *(newsolution->myproblem->returnrequest(rand1));
    request request2=*(newsolution->myproblem->returnrequest(rand2));

    if(!newsolution->is_requestssameroute(request1,request2))
        newsolution->merge3(request1,request2);

    if(newsolution->returnroutenumber()==k-1)
    {
        solution* copy=new solution(myproblem);
        copy_newsolution->assignsolution(copy);
        solutionpool.push_back(copy);
        k=k-1;
    }
    newsolution->assignsolution(copy_newsolution);
    solutionpool.push_back(newsolution);
}

for (int i=0;i<solutionpool.size();i++)
    solutionpool[i]->writetofile();
return solutionpool;
}

solution* solution::variatesolution2()
{
    start:

    int rand1= rand()%(myproblem->readrequestsize());

```

```

int rand2= rand()%(myproblem->readrequestsiz());

if(rand1==rand2)
    goto start;

request request1= *(myproblem->returnrequest(rand1));
request request2=*(myproblem->returnrequest(rand2));

//if two chosen requests are on same route then exchange, else merge
if((is_requestssameroute(request1,request2)))
    exchange_for_same_destination(request1,request2);
else if((merge(request1,request2))==0)
{
    exchange2_for_same_destination(request1,request2);
}
solution* newsolution=new solution(myproblem);
assignsolution(newsolution);

return newsolution;
}

```

//The modification for the service problem

```
void solution::createinitialroutes()
```

```

{    // Each request is assigned to a vehicle

    int size=myproblem->readvehiclesize();
    start:

    vector<int> list;
    for(int i=0;i<size;i++)
        list.push_back(i);

    // Shuffle

    for (int x=size;x>1;x--)

```

```

{
    int rand_ind=rand()%x;
    int temp=list[rand_ind];
    list[rand_ind]=list[x-1];
    list[x-1]=temp;
}
int k=0;
int it=0;

for(;it<size;it++)
{
    myroutes.push_back(new route(myproblem));
    vehicle* veh;
    veh=myproblem->returnvehicle(list[k]);
    myroutes[it]->assign_vehicle_from_myvehicles(veh);
    point* sourcepoint= myproblem->returnsource(it);
    point* destinationpoint=myproblem->returndestination(it);
    myroutes[it]->addpointtoroute(sourcepoint);
    myroutes[it]->addpointtoroute(destinationpoint);
    k++;
}

for(;it<myproblem->readrequestsize();it++)
{
    int index=rand()%size;
    vector<point*>::iterator iter;
    iter=myroutes[index]->returnmypoints().end();
    myroutes[index]->addpointtoroute(myproblem->returnsource(it));
}
convert(myroutes);
if(is_feasible()!=1)
    goto start;
writetofile();
writefxtofile();
}
/*
void solution::createinitialroutes()

```

```

{    // Each request is assigned to a vehicle
    int size=myproblem->readvehiclesize();
    start:
    vector<int> list;
    for(int i=0;i<size;i++)
        list.push_back(i);
    // Shuffle
    for (int x=size;x>1;x--)
    {
        int rand_ind=rand()%x;
        int temp=list[rand_ind];
        list[rand_ind]=list[x-1];
        list[x-1]=temp;
    }
    int k=0;

    for(int it=0;it<size;it++)
    {
        //int a=rand()%size;
        myroutes.push_back(new route(myproblem));
        vehicle* veh;
        veh=myproblem->returnvehicle(list[k]);
        myroutes[it]->assign_vehicle_from_myvehicles(veh);
        point* sourcepoint= myproblem->returnsource(it);
        point* destinationpoint=myproblem->returndestination(it);
        myroutes[it]->addpointtoroute(sourcepoint);
        myroutes[it]->addpointtoroute(destinationpoint);
        /*if(myroutes[it]->is_feasible()!=1)
        {
            myroutes[it]->writeroute();
            goto start;
        }
        else
            k++;
    }
    if(is_feasible()!=1)

```

```

        goto start;

    writetofile();
    writefxtofile();
}
*/
bool arethesame(point* a, point* b)
{
    if(a->returnx()==b->returnx() && a->returny()==b->returny()) return 1;
    else return 0;
}

void solution::exchange2_for_same_destination(request request1,request request2)
{
    if(arethesame(request1.readdestination(),request2.readdestination())==1)
    {
        exchange2(request1, request2);
    }
}

void solution::exchange2(request request1,request request2)
{
    int request_route_id1;
    int request_route_id2;

    request_route_id1=return_request_of_route_id(request1);
    request_route_id2=return_request_of_route_id(request2);

    //The source and destination point of request2 are replaced in request1 in place of
    //the source and destination point of request1

    route* copy_route1=new route(myproblem);
    *copy_route1=*myroutes[request_route_id1];

    route* copy_route2=new route(myproblem);
    *copy_route2=*myroutes[request_route_id2];

    for(int i=0;i<(myroutes[request_route_id1]->returnsizeofpoints());i++)

```

```

        if(myroutes[request_route_id1]->returnpoint(i)==request1.readsource())
            copy_route1->change_points(i,request2.readsource());
        else if(myroutes[request_route_id1]->returnpoint(i)==request1.readdestination())
            copy_route1->change_points(i,request2.readdestination());

//The source and destination point of request2 are replaced in request1 in place of
//the source and destination point of request1

for(int i=0;i<(myroutes[request_route_id2]->returnsizeofpoints());i++)
    if(myroutes[request_route_id2]->returnpoint(i)==request2.readsource())
        copy_route2->change_points(i,request1.readsource());
    else if(myroutes[request_route_id2]->returnpoint(i)==request2.readdestination())
        copy_route2->change_points(i,request1.readdestination());

//feasibility check after exchange
if(copy_route1->is_feasible()&& copy_route2->is_feasible())
{
    myroutes[request_route_id1]=copy_route1;
    myroutes[request_route_id2]=copy_route2;
}
}

void solution::exchange_for_same_destination(request request1,request request2)
{
    if(arethesame(request1.readdestination(),request2.readdestination())==1)
        { exchange(request1, request2); }
}

void solution::exchange(request request1,request request2)
{
    int request_route_id;

    request_route_id=return_request_of_route_id(request1);

    route* copy_route=new route(myproblem);
    *copy_route=*myroutes[request_route_id];

//The source and destination point of request2 are replaced in request1 in place of

```

```

//the source and destination point of request1

for(int i=0;i<(myroutes[request_route_id]->returnsizeofpoints());i++)
    if(myroutes[request_route_id]->returnpoint(i)==request1.readsource())
        copy_route->change_points(i,request2.readsource());
    else if(myroutes[request_route_id]->returnpoint(i)==request1.readdestination())
        copy_route->change_points(i,request2.readdestination());

//The source and destination point of request1 are replaced in request1 in place of
//the source and destination point of request2

for(int i=0;i<(myroutes[request_route_id]->returnsizeofpoints());i++)
    if(myroutes[request_route_id]->returnpoint(i)==request2.readsource())
        copy_route->change_points(i,request1.readsource());
    else if(myroutes[request_route_id]->returnpoint(i)==request2.readdestination())
        copy_route->change_points(i,request1.readdestination());

//feasibility after exchange
if(copy_route->is_feasible())
    myroutes[request_route_id]=copy_route;
}

void solution::merge3(request request1,request request2)
{
    if(arethesame(request1.readdestination(),request2.readdestination())==1)
        { merge2(request1, request2); }
}

void solution::merge2(request request1,request request2)
{
    double totalfx1=0;
    double totalfx2=0;
    route* bestroute1=new route(myproblem);
    route* bestroute2=new route(myproblem);

    route* route1=returnroute(return_request_of_route_id(request1));
    route* route2=returnroute(return_request_of_route_id(request2));

```

```

vector<route*>routelist1;
vector<route*>routelist2;

vector<point*>::iterator iter1,iter2,iter3,iter4;

int i=1,j=0;

int cont1=1;
//The replacement of request1 in the route of request2
for(iter1=route2->returnbegin();cont1==1;){
    int cont2=1;
    for(iter2=iter1;cont2==1;)
    {
        route* newroute=new route(myproblem);
        *newroute=*route2;

        iter3=newroute->returnbegin()+j;

        newroute->add(iter3,request1.readsource());
        if(newroute->is_feasible())
            routelist1.push_back(newroute);
        else delete newroute;
        if(iter2==route2->returnend())    cont2=0;
        if(cont2==1)    iter2++;
        ++i;
    }
    if(iter1==route2->returnend())    cont1=0;
    if(cont1==1)    iter1++;
    i=1;
    j++;
}
if(routelist1.size()!=0)
{
    int bestindex1=0;
    for(int i=1;i<routelist1.size();i++)
        if(routelist1[i]->returnfx()<routelist1[bestindex1]->returnfx())

```

```

        bestindex1=i;
    bestroute1=routelist1[bestindex1];

    if((route1->returnsizeofpoints()-2)==0)
        totalfx1=bestroute1->returnfx();
    else
    {
        route aroute2=*route1;
        aroute2.changemypoints(route1->eraserequest(request1.readsource()));
        route aroute3=aroute2;
        totalfx1=bestroute1->returnfx()+aroute3.returnfx(); }
}

i=1;
j=0;

int cont3=1;

//The replacement of request2 in the route of request1
for(iter1=route1->returnbegin();cont3==1;){
    int cont4=1;
    for(iter2=iter1;cont4==1;)
    {
        route* newroute=new route(myproblem);
        *newroute=*route1;
        iter3=newroute->returnbegin()+j;
        newroute->add(iter3,request2.readsource());
        if(newroute->is_feasible())
            routelist2.push_back(newroute);
        else delete newroute;
        ++i;
        if(iter2==route1->returnend())    cont4=0;
        if(cont4==1)    iter2++;
    }
    if(iter1==route1->returnend())    cont3=0;
    if(cont3==1)    iter1++;

    i=1;

```

```

        j++;
    }
    if(routelist2.size()!=0)
    {
        int bestindex2=0;
        for(int i=1;i<routelist2.size();i++)
            if(routelist2[i]->returnfx()<routelist2[bestindex2]->returnfx())
                bestindex2=i;
        *bestroute2=*routelist2[bestindex2];
        if((route2->returnsizeofpoints()-2)==0)
            totalfx2=bestroute2->returnfx();
        else
        {
            route aroute=*route2;
            aroute.changemypoints(route2->eraserequest(request2.readsource()));
            route aroute4=aroute;
            totalfx2=bestroute2->returnfx()+aroute4.returnfx();
        }
    }
    if(routelist1.size()!=0 && routelist2.size()!=0)
    {
        if(totalfx1<=totalfx2)
        {
            if(route1->returnsizeofpoints()-2==0)
                myroutes.erase(findrouteit(route1));
            else
            {
                route route5=*route1;
                route5.changemypoints(route1->eraserequest(request1.readsource()));
                route route6=route5;
                route1->changemypoints(route6.returnmypoints());
            }

            *route2=*bestroute1;
        }
        else
        {

```

```

        if(route2->returnsizeofpoints()-2==0)
            myroutes.erase(findrouteit(route2));
        else
        {
            route route7=*route2;
            route7.changemypoints(route2->eraserequest(request2.readsource()));
            route route8=route7;
            route2->changemypoints(route8.returnmypoints());
        }
        *route1=*bestroute2;
    }
}

if(routelist1.size()==0 && routelist2.size()!=0)
{
    if(route2->returnsizeofpoints()-2==0)
        myroutes.erase(findrouteit(route2));
    else
    {
        route route7=*route2;
        route7.changemypoints(route2->eraserequest(request2.readsource()));
        route route8=route7;
        route2->changemypoints(route8.returnmypoints());
    }
    *route1=*bestroute2;
}

if(routelist2.size()==0 && routelist1.size()!=0)
{
    if(route1->returnsizeofpoints()-2==0)
        myroutes.erase(findrouteit(route1));
    else
    {
        route route5=*route1;
        route5.changemypoints(route1->eraserequest(request1.readsource()));
        route route6=route5;
        route1->changemypoints(route6.returnmypoints());
    }
}

```

```

        *route2=*bestroute1;
    }
    for(int i=0;i<routelist1.size();i++)
    { delete routelist1[i]; }

    for(int i=0;i<routelist2.size();i++)
    { delete routelist2[i]; }
}

bool solution::merge(request request1,request request2)
{
    if(arethesame(request1.readdestination(),request2.readdestination())==0)
    { return 0; }
    double totalfx1=0;
    double totalfx2=0;
    route* bestroute1=new route(myproblem);
    route* bestroute2=new route(myproblem);

    route* route1=returnroute(return_request_of_route_id(request1));
    route* route2=returnroute(return_request_of_route_id(request2));

    vector<route*>routelist1;
    vector<route*>routelist2;

    vector<point*>::iterator iter1,iter2,iter3,iter4;

    int i=1,j=0;

    int cont1=1;
    //The replacement of request1 in the route of request2
    for(iter1=route2->returnbegin();cont1==1;){
        int cont2=1;
        for(iter2=iter1;cont2==1;)
        {
            route* newroute=new route(myproblem);
            *newroute=*route2;

```

```

        iter3=newroute->returnbegin()+j;
        newroute->add(iter3,request1.readsource());
        if(newroute->is_feasible())
            routelist1.push_back(newroute);
        else delete newroute;
        if(iter2==route2->returnend())    cont2=0;
        if(cont2==1)    iter2++;

        ++i;
    }
    if(iter1==route2->returnend())    cont1=0;
    if(cont1==1)    iter1++;
    i=1;
    j++;
}
if(routelist1.size()!=0)
{
    int bestindex1=0;
    for(int i=1;i<routelist1.size();i++)
        if(routelist1[i]->returnfx()<routelist1[bestindex1]->returnfx())
            bestindex1=i;
    bestroute1=routelist1[bestindex1];

    if((route1->returnsizeofpoints()-2)==0)
        totalfx1=bestroute1->returnfx();
    else
    {
        route aroute2=*route1;
        aroute2.changemypoints(route1->eraserequest(request1.readsource()));
        route aroute3=aroute2;
        totalfx1=bestroute1->returnfx()+aroute3.returnfx();
    }
}
i=1;
j=0;

int cont3=1;

```

```

//The replacement of request2 in the route of request1
for(iter1=route1->returnbegin();cont3==1;){
    int cont4=1;
    for(iter2=iter1;cont4==1;)
    {
        route* newroute=new route(myproblem);
        *newroute=*route1;
        iter3=newroute->returnbegin()+j;

        newroute->add(iter3,request2.readsource());

        if(newroute->is_feasible())
            routelist2.push_back(newroute);
        else delete newroute;
        ++i;
        if(iter2==route1->returnend())    cont4=0;
        if(cont4==1)    iter2++;
    }
    if(iter1==route1->returnend())    cont3=0;
    if(cont3==1)    iter1++;
    i=1;
    j++;
}

if(routelist2.size()!=0)
{
    int bestindex2=0;
    for(int i=1;i<routelist2.size();i++)
        if(routelist2[i]->returnfx()<routelist2[bestindex2]->returnfx())
            bestindex2=i;
    *bestroute2=*routelist2[bestindex2];
    if((route2->returnsizeofpoints()-2)==0)
        totalfx2=bestroute2->returnfx();
    else
    {
        route aroute=*route2;

```

```

        aroute.changemypoints(route2->eraserequest(request2.readsource()));
        route aroute4=aroute;
        totalfx2=bestroute2->returnfx()+aroute4.returnfx();
    }
}
if(routelist1.size()!=0 && routelist2.size()!=0)
{
    if(totalfx1<=totalfx2)
    {
        if(route1->returnsizeofpoints()-2==0)
            myroutes.erase(findrouteit(route1));
        else
        {
            route route5=*route1;
            route5.changemypoints(route1->eraserequest(request1.readsource()));
            route route6=route5;
            route1->changemypoints(route6.returnmypoints());
        }

        *route2=*bestroute1;
    }
    else
    {
        if(route2->returnsizeofpoints()-2==0)
            myroutes.erase(findrouteit(route2));
        else
        {
            route route7=*route2;
            route7.changemypoints(route2->eraserequest(request2.readsource()));
            route route8=route7;
            route2->changemypoints(route8.returnmypoints());
        }
        *route1=*bestroute2;
    }
}

if(routelist1.size()==0 && routelist2.size()!=0)

```

```

{
    if(route2->returnsizeofpoints()-2==0)
        myroutes.erase(findrouteit(route2));
    else
    {
        route route7=*route2;
        route7.changemypoints(route2->eraserequest(request2.readsource()));
        route route8=route7;
        route2->changemypoints(route8.returnmypoints());
    }
    *route1=*bestroute2;
}

if(routelist2.size()==0 && routelist1.size()!=0)
{
    if(route1->returnsizeofpoints()-2==0)
        myroutes.erase(findrouteit(route1));
    else
    {
        route route5=*route1;
        route5.changemypoints(route1->eraserequest(request1.readsource()));
        route route6=route5;
        route1->changemypoints(route6.returnmypoints());
    }
    *route2=*bestroute1;
}

for(int i=0;i<routelist1.size();i++)
{ delete routelist1[i]; }

for(int i=0;i<routelist2.size();i++)
{ delete routelist2[i]; }

if(routelist1.size()==0 && routelist2.size()==0) return 0;
else return 1;

}

```

```

void solution::assignsolution(solution* asolution)
{
    //asolution->myproblem=this->myproblem;
    asolution->myroutes.clear();

    for(int i=0;i<myroutes.size();i++)
        asolution->myroutes.push_back(new route(myproblem));

    for(int i=0;i<myroutes.size();i++)
        *asolution->myroutes[i]=*myroutes[i];
}

void solution::assignsolution(solution asolution)
{
    asolution.myroutes.clear();

    for(int i=0;i<myroutes.size();i++)
        asolution.myroutes.push_back(new route(myproblem));

    for(int i=0;i<myroutes.size();i++)
        *asolution.myroutes[i]=*myroutes[i];
}

void solution::write()
{
    FILE* cfPtr;
    if((cfPtr=fopen("data.txt","a"))==NULL)
        cout<<"The file can not be opened"<<endl;
    else{
        fprintf(cfPtr,"\n\n");
        fprintf(cfPtr,"%50s","SA SOLUTION");
    }
    fclose(cfPtr);
    writetofile();
    writefxtofile();
}

```

```

vector<solution*> solution::solve()
{
    createinitialroutes();
    //convert(myroutes);

    solution* keepit=new solution(myproblem);
    assignsolution(keepit);

    solution* SASolution=new solution(myproblem);
    //SASolution=solvewithSA();

    vector<solution*> solutionpool(variatesolution());
    // SASolution=solvewithSA2(solutionpool);
    return solutionpool;
}

bool solution::is_requestfeasible()
{
    int result=1;
    int total=0;
    for(int a=0;a<myroutes.size();a++)
    {
        total=total+myroutes[a]->returnsizeofpoints();
    }
    if(total!=2*(myproblem->readrequestsize()))result=0;
    return result;
    /*
    bool result=1;
    for(int i=0;i<myproblem->readrequestsize()&&result==1;i++)
        for(int j=0;j<myroutes.size();j++)
            for(int k=0;k<myroutes[j]->returnsizeofpoints();k++){
                if((myproblem->returnrequest(i))->readsource()==myroutes[j]-
>returnpoint(k))
                    result= 1;
            }
    }
    */
}

```

```

        else
            result=0;
    }

    return result;
    */
}

bool solution::is_routefeasible()
{
    bool result=1;
    for(int a=0;a<myroutes.size()&&result==1;a++)
    { if(myroutes[a]->is_feasible()==0) result=0; }
    return result;
}

bool solution::is_requestssameroute(request first, request second)
{
    int route1;
    int route2;

    bool result=1;
    for(int i=0;i<myroutes.size();i++){
        for(int j=0;j<myroutes[i]->returnsizeofpoints();j++){
            if(myroutes[i]->returnpoint(j)==first.readsource())
                route1=i;
            else if (myroutes[i]->returnpoint(j)==second.readsource())
                route2=i;}

        if(route1!=route2)
            result=0;
    }
    return result;
}

bool solution::is_feasible()
{
    bool result;
    result = is_routefeasible();/*is_requestfeasible();

```

```

        return result;
    }

void solution::print_total_fx()
{ cout<<total_fx<<endl; }

//The modification for the service problem
void solution::writetofile()
{
    convert(myroutes);
    FILE* cfPtr;
    if((cfPtr=fopen("data.txt","a"))==NULL)
        cout<<"The file can not be opened"<<endl;
    else
    {
        fprintf(cfPtr, "\n%80s\n\n", "SOLUTION");
        fprintf(cfPtr, "%35s%25s%", "Vehicle Number", "Points");

        for(int a=0;a<myroutes.size();a++){
            fprintf(cfPtr, "\n");
            for(int i=0;i<myproblem->readvehiclesize();i++){
                if((myroutes[a]->returnvehicle())==myproblem->returnvehicle(i))
                    fprintf(cfPtr, "%-s%-15d%-20d", "Route ", a, i);
            }
            for(int j=0;j<myroutes[a]->returnsizeofpoints();j++)
                for(int i=0;i<myproblem->readpointssize();i++)
                    if(myroutes[a]->returnpoint(j)==myproblem->returnpoint(i))
                        fprintf(cfPtr, "%s%-10d", "Point ", i);
            }
        }
    fclose(cfPtr);
}

/*
void solution::writetofile()
{

```

```

FILE* cfPtr;
if((cfPtr=fopen("data.txt","a"))==NULL)
    cout<<"The file can not be opened"<<endl;
else
{
    fprintf(cfPtr, "\n%80s\n\n", "SOLUTION");
    fprintf(cfPtr, "%35s%25s%140s\n", "Vehicle Number", "Points", "Is route feasible?");

    for(int a=0;a<myroutes.size();a++){
        fprintf(cfPtr, "\n");
        for(int i=0;i<myproblem->readvehiclesize();i++){
            if((myroutes[a]->returnvehicle())==myproblem->returnvehicle(i))
                fprintf(cfPtr, "%s%-15d%-20d", "Route ", a, i);
        }
        for(int j=0;j<myroutes[a]->returnsizeofpoints();j++)
            for(int i=0;i<myproblem->readpointssize();i++)
                if(myroutes[a]->returnpoint(j)==myproblem->returnpoint(i))
                    fprintf(cfPtr, "%s%-10d", "Point ", i);

        if(myroutes[a]->is_feasible())
            fprintf(cfPtr, "%122d", 1);
        else
            fprintf(cfPtr, "%122d", 0);
    }
    fclose(cfPtr);
}

*/

void manager::run()
{
    myproblem=createinstance(/*0,*/vehicle_size,request_size);
    mysolution=new solution(myproblem);
    algorithm* myalgorithm=new algorithm;
    myalgorithm->solve(mysolution);
}

```

```
problem* manager::createinstance(/*int seed,*/int number_of_vehicles,int number_of_requests)
{ return new problem(number_of_vehicles,number_of_requests); }
```

```
point::point(double a, double b)
{
    x=a;
    y=b;
    myrequest=NULL;
}
```

```
void point::definerequest(request* owner)
{ myrequest=owner; }
```

```
point::point()
    { myrequest=NULL; }
void point::set_is_depot(int a)
{ is_depot=a; }
```

```
void point::random_generation()
{
    x=(double)rand()/RAND_MAX*100;
    y=(double)rand()/RAND_MAX*100;
}
```

```
void point::setdatamembers(double lb, double ub, double l, double st)
{
    earliestarrival=lb;
    latestarrival=ub;
    load=l;
    servicetime=st;
}
```

```
void point::setx(double a)
{ x=a; }
void point::sety(double a)
{ y=a; }
```

```
double point::whatisyourload()
```

```

{ return load; }

double point::returnearliestarrival()
{ return earliestarrival; }

double point::returnlatestarrival()
{ return latestarrival; }

double point::returnservicetime()
{ return servicetime; }

double point::returnx()
{ return x; }

double point::returny()
{ return y; }

request::request()
{
    d=0;
    double aa,bb,cc,dd;
    aa=data_source_xy[c];
    c=c+1;
    bb=data_source_xy[c];
    c=c+1;
    cc=data_destination_xy[d];
    d=d+1;
    dd=data_destination_xy[d];

    //source=new point((double)rand()/RAND_MAX*100,(double)rand()/RAND_MAX*100);
    source=new point(aa,bb);

    //destination=new point((double)rand()/RAND_MAX*100,(double)rand()/RAND_MAX*100);
    destination=new point(cc,dd);

    //load=(double)rand()/RAND_MAX*100;

    load=data_load[e];
    e=e+1;

    //servicetime=(double)rand()/RAND_MAX*10;

```

```

servicetime=data_servicetime[f];
f=f+1;

//earliestpickup=(double)rand()/RAND_MAX*600;
earliestpickup=0;
//latestpickup=(double)rand()/RAND_MAX*600;
latestpickup=50;
//earliestdelivery=(double)rand()/RAND_MAX*600;
earliestdelivery=0;
//latestdelivery=(double)rand()/RAND_MAX*600;
latestdelivery=50;
//latestpickup=earliestpickup;
/*for(;latestpickup<=earliestpickup;) latestpickup=(double)rand()/RAND_MAX*200;
earliestdelivery=earliestpickup;
for(;earliestdelivery<=earliestpickup;) earliestdelivery=(double)rand()/RAND_MAX*200;
latestdelivery=earliestdelivery;
for(;latestdelivery<=earliestdelivery          ||          latestdelivery<=latestpickup;)
latestdelivery=(double)rand()/RAND_MAX*200;*/
source->setdatamembers(earliestpickup,latestpickup,load,servicetime);
destination->setdatamembers(earliestdelivery,latestdelivery,-load,servicetime);
}

point* request::readsource()
{ return source; }
point* request::readdestination()
{ return destination; }

void main()
{
    srand(0);
    int i;
    clock_t start,end;
    double elapsed;
    start=clock();
    vector<double>::iterator endIterator;

    FILE *a;
```

```

if((a=fopen("Source.txt","r+"))==NULL)
    cout<<"Dosya acilamadi\n";
else
{
    while(!feof(a))
    {
        double x;
        fscanf(a,"%lf",&x);
        data_source_xy.push_back(x);
    }
}
fclose(a);

endlIterator=data_source_xy.end();
data_source_xy.erase(endlIterator-1);

FILE *b;
if((b=fopen("Destination.txt","r+"))==NULL)
    cout<<"Dosya acilamadi\n";
else
{
    while(!feof(b))
    {
        double x;
        fscanf(b,"%lf",&x);
        data_destination_xy.push_back(x);
    }
}
fclose(b);

endlIterator=data_destination_xy.end();
data_destination_xy.erase(endlIterator-1);

FILE *c;
if((c=fopen("Service.txt","r+"))==NULL)
    cout<<"Dosya acilamadi\n";
else

```

```

{
    while(!feof(c))
    {
        double x;
        fscanf(c, "%lf", &x);
        data_servicetime.push_back(x);
    }
}
fclose(c);

endIterator=data_servicetime.end();
data_servicetime.erase(endIterator-1);

FILE *e;
if((e=fopen("Capacity.txt", "r+"))==NULL)
    cout<<"Dosya acilamadi\n";
else
{
    while(!feof(e))
    {
        double x;
        fscanf(e, "%lf", &x);
        data_capacity.push_back(x);
    }
}
fclose(e);

endIterator=data_capacity.end();
data_capacity.erase(endIterator-1);

FILE *f;
if((f=fopen("Load.txt", "r+"))==NULL)
    cout<<"Dosya acilamadi\n";
else
{
    while(!feof(f))
    {

```

```

        double x;
        fscanf(f,"%lf",&x);
        data_load.push_back(x);
    }
}
fclose(f);

endIterator=data_load.end();
data_load.erase(endIterator-1);

manager* mymanager=new manager;
mymanager->run();
end=clock();
elapsed=((double)(end-start))/CLOCKS_PER_SEC;
FILE* cfPtr;
cfPtr=fopen("data.txt","a");
fprintf(cfPtr,"\n\n");
fprintf(cfPtr,"%s%.3f\n","Elapsed: ",elapsed);
fclose(cfPtr);
cin>>i;
}

```

REFERENCES

1. Altmel, İ. K. and T. Öncan, 2005, "New enhancement of Clarke and Wright savings heuristic for the capacitated vehicle routing problem", *Journal of the Operational Society*, Vol. 8, pp. 954-961.
2. Beasley, J. E., 1983, *Route- first cluster-second methods for vehicle routing*. Omega, Vol. 11, pp. 403-408.
3. Bent, R. and P. V. Hentenryck, 2004, "A two-stage hybrid algorithm for the vehicle routing problem with time windows", *Transportation Science*, Vol. 38 (4), pp. 515-530.
4. Bent, R. and P. V. Hentenryck, 2006, "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows", *Computers and Operations Research*, Vol. 33, pp. 875-893.
5. Berbeglia, G., JF. Cordeau, I. Gribkovskaia and G. Laporte, 2007, *Static pickup and delivery problems: A classification scheme and survey*. TOP, Vol. 15, pp. 1-31.
6. Bodin, L. D., B. L. Golden, A. A. Assad and M. Ball, 1983, "Routing and scheduling of vehicles and crews", *Computers and Operations Research*, Vol. 10, pp. 63-212.
7. Bodin, L. D. and T. Sexton, 1986, "The multi-vehicle subscriber dial-a-ride problem", *TIMS Studies in Management Science*, Vol. 26, pp. 73-86.
8. Bramel, J. and D. Simchi-Levi, 1995, "A location based heuristic for general routing problems", *Operations Research*, Vol. 43, pp. 649-660.
9. Breedam, A. V., 1994 , *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*, Ph.D. dissertation, University of Antwerp.

10. Cerny, V., 1985, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm", *Journal of Optimization Theory Application*, Vol. 45, pp. 41-51.
11. Chiang, W. and R. Russell, 1996, "Simulated annealing metaheuristics for the vehicle routing problem with time windows", *Ann. Oper. Res.*, Vol. 63, pp. 3-27.
12. Chiang, W. and R. Russell, 1997, "A reactive tabu search metaheuristic for the vehicle routing problem with time windows", *INFORMSJ. Comput.*, Vol. 9, pp. 417-430.
13. Christofides, N., A. Mingozzi and P. L. Toth, 1979, *The vehicle routing problem*, In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*. Wiley, Chichester, UK, pp. 315-318.
14. Reeves, C. R., 1993, *Modern heuristic techniques for combinatorial problems*, pp. 1-55.
15. Cook, S. A., 1971, *The complexity of theorem-providing procedures*, Proceedings of the 3rd Annual ACM Symposium on the theory of computing, pp. 151-158.
16. Cordeau, J., G. Laporte and A. Mercier, 2001, "A unified tabu search heuristic for vehicle routing problems with time windows", *J. Oper. Res. Soc.*, Vol. 52, pp. 928-936.
17. Cordeau, J. F. and G. Laporte, 2003, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem", *Transportation Research B*, Vol. 37, pp. 579-594.
18. De Backer, B., V. Furnon, P. Shaw, P. Kilby and P. Prosser, 2000, *Solving vehicle routing problems using constraint programming and metaheuristics*, *J. Heuristics*, Vol. 6, pp. 501-523.
19. Derigs, U. and T. Döhmer, 2006, "Indirect search for the vehicle routing problem with pickup and delivery and time windows", *OR Spectrum*.

20. Dessouky, M. M., M. Rahimi and M. Weidner, 2003, "Jointly optimizing cost, service, and, environmental performance in demand-responsive transit scheduling", *Transportation Research Part D: Transport and Environment*, Vol. 8, pp. 433-465.
21. Fisher, M. L. and R. Jaikumar, 1981, "A generalized assignment heuristic for the vehicle routing problem". *Networks*, Vol. 11, pp. 109-124.
22. Fisher, M. L., In O.M. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser, 1995, "Vehicle Routing", Network Routing, *Handbooks in Operations Research and Management Science*, Vol. 8, North-Holland, Amsterdam, pp. 1-33.
23. Gambardella, L., E. Taillard and G. Agazzi, 1999, "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows", D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization*. McGraw-Hill, London, U.K., pp. 63-76.
24. Garey, M. R. and D. S. Johnson, 1979, *Computers and Intractability: A guide to the theory of NP-completeness*, W.F. Freeman and Company, New York.
25. Gaskell, T. J., 1967, "Bases for vehicle fleet scheduling", *Operational Research Quarterly*, Vol. 18, pp. 281-295.
26. Gehring, H. and J. Homberger, 1999, *A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows*, Proc. EUROGEN99—Short Course on Evolutionary Algorithms Engrg. Comput. Sci., Reports of the Department of Mathematical Information Technology Series A, Collections, A2/1999, University of Jyväskylä, Jyväskylä, Finland, pp. 57-64.
27. Gendreau, M., G. Laporte and J. Potvin, 1997, "Vehicle routing: Modern heuristics", E. Aarts, J. Lenstra, eds. *Local Search in Combinatorial Optimization*, Ch. 9. JohnWiley & Sons Ltd., New York, pp. 311-336.
28. Gillett, B. E. and L. R. Miller, 1974, "A heuristic algorithm for the vehicle dispatch

- problem”, *Operations Research*, Vol. 22, pp. 340-349.
29. Golden, B. L. and A. A. Assad, 1998, *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam.
 30. Gribkovskaia, I., O. Halskau, G. Laporte and M. Vlcek, 2007, “General solutions to the single vehicle routing problem with pickups and deliveries”, *European Journal of Operational Research*, Vol. 180, pp. 568-584.
 31. Ioachim, I., J. Desrosiers, Y. Dumas, M. M. Solomon and D. Villeneuve, 1995, “A request clustering algorithm for door-to-door handicapped “transportation”, *Transportation Science*, Vol. 29 (1), pp. 63-78.
 32. Jaw, J. J., A. R. Odoni, H. N. Psaraftis and N. H. M. Wilson, 1986, “A heuristic algorithm for the multi vehicle advance-request dial-a-ride problem with time windows”, *Transportation Research B*, Vol. 20, pp. 243-257.
 33. Johnson, D., C. Aragon, L. McGeoch and C. Schevon, 1989, “Optimization by simulated annealing: an experimental evaluation; part i: Graph partitioning”, *Operations Research*, Vol. 37, pp. 865-92.
 34. Jüngen, S. and F. Torsten, 1997, *A parallel algorithm for the vehicle routing problem with time window constraints*, Department of Computer Science, University of Paderborn.
 35. Kindervater, G. A. P. and M. W. P. Savelsbergh, 1997, “Vehicle Routing: Handling edge exchanges”, In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, Wiley, Chicester, UK, pp. 337-360.
 36. Kirkpatrick, S., D. C. Gellat and M. P. Vecchi, 1983, “Optimization by simulated annealing”, *Science*, Vol. 220, pp. 671-680.
 37. Landrieu, A., Y. Mati and Z. Binder, 2001, “A tabu search heuristic for the single vehicle pickup and delivery problem with time windows”, *Journal of Intelligent*

Manufacturing, Vol. 12, pp. 497-508.

38. Lao, H. C. and Z. Liang, 2002, "Pickup and delivery problem with time windows: Algorithms and test case generation", *International Journal on Artificial Intelligence Tools* (Architectures, Languages, Algorithms), Vol. 11 (3), pp. 455-472.
39. Li, H. and A. Lim, 2001, *A metaheuristic for the pickup and delivery problem with time windows*, In: Proceedings 13th IEEE ICTAI 2001, Los Alamitos, CA., pp. 160-167.
40. Lin, S., 1965, "Computer solutions of the traveling salesman problem", *Bell System Technical Journal*, Vol. 44, pp. 2245-2269.
41. Lu, Q. and M. M. Dessouky, 2006, "A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows", *European Journal of Operational Research*, Vol. 175, pp. 672-687.
42. Madsen, O. B. G., H. F. Ravn and J. M. Rygaard, 1995, "A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives", *Annals of Operations Research*, Vol. 60, pp. 193-208.
43. Mauri, G. R. and L. A. N. Lorena, 2006, *A multiobjective model and simulated annealing approach for a dial-a-ride problem*, Laboratory for Computing and Applied Mathematics -National Institute for Space Research.
44. Melachrinoudis, E., A. B. Ilhan and H. Min, 2007, "A dial-a-ride problem for client transportation in a health-care organization", *Computers & Operations Research*, Vol. 34, pp. 742-759.
45. Metropolis, N., A. W., Rosenbluth, M. N., Rosenbluth, A. H., Teller, and E., Teller, 1953, "Equation of state calculation by fast computing machines", *J. Of Chem. Phys.*, Vol. 21, pp. 1087-1091.
46. Mole, R. H., and S. R., Jameson, 1976, "A sequential route-building algorithm

- employing a generalized savings criterion”, *Operatioal Research Quarterly*, Vol. 27, pp. 503-511.
47. Nagy, G. and S. Salhi, 2005, “Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries”, *European Journal of Operational Research*, Vol. 162, pp. 126-141.
 48. Nanry, W. P. and J. W. Barnes, 1998, “Solving the pickup and delivery problem with time Windows using reactive tabu search”, *Transportation Research B*, Vol. 34 (2), pp. 107-121.
 49. 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.
 50. Osman, I. H. and J. P. Kelly, 1996, *Meta-heuristics: Theory&Applications*, Kluwer Academic Publishers, Boston, London .
 51. Özdamar, L. and G. Barbarasoglu, 2000, *Int. J. Production Economics* 68, pp. 319-331.
 52. Paolo, T. and V. Daniele, 1997, ”Heuristic algorithm for the handicapped persons transportation problem”, *Transportation Science*, Vol. 31 (1), pp. 60-71.
 53. Paolo, T. and V. Daniele, 2001, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
 54. Parragh, S. N., K. F. Doerner and R. F. Hartl, 2006, “A suvery on pickup and delivery problems. Part II: Transportation between pickup ad delivery locations”, State-of-the-art-artikel.
 55. Potvin, J. and S. Begio, 1996, “The vehicle routing problem with time windows—Part II: Genetic search”, *INFORMS J. Comput.*, Vol. 8, pp. 165–172.

56. Rochat, Y. and E. Taillard, 1995, "Probabilistic diversification and intensification in local search for vehicle routing", *J. Heuristics*, Vol. 1, pp. 147–167.
57. Rousseau, L., M. Gendreau and G. Pesant, 2002, "Using constraint-based operators to solve the vehicle routing problem with time windows", *J. Heuristics*, Vol. 8, pp. 43–58.
58. Shaw, P., 1998, Using constraint programming and local search methods to solve vehicle routing problems, Proc. Principles Practice Constraint Programming, Pisa, Italy, pp. 417–431.
59. Taillard, E., P. Badeau, M. Gendreau, F. Geurtin and J. Potvin, 1997, "A tabu search heuristic for the vehicle routing problem with soft time windows", *Transportation Sci.*, Vol. 31 170–186.
60. Tan^{a*}, K.C., L.H. Lee^b, Q.L. Zhu^a and K. Qu^a, 2000, "Heuristic methods for vehicle routing problem with time windows", *Artificial Intelligence in Engineering*, Vol.15, pp. 281-295.
61. Thompson, P. M. and H. N. Psaraftis, 1993, "Cyclic transfer algorithms for multi-vehicle routing and scheduling problems", *Operations Research*, Vol. 41, pp. 935-946.
62. Van der Bruggen, L. J. J., J. K. Lenstra, P. C. Schuur, 1993, "Variable-depth search for the single vehicle pickup and delivery problem with time windows", *Transportation Science*, Vol. 27, pp. 298-311.
63. Yellow, P., 1970, "A computational modification to the savings method of vehicle scheduling", *Operational Research Quarterly*, Vol.21, pp. 281-283.