

AN APPLICATION OF THE VEHICLE ROUTING PROBLEM TO A
GLASS MANUFACTURING FIRM

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

BY

İPEK SEYRAN

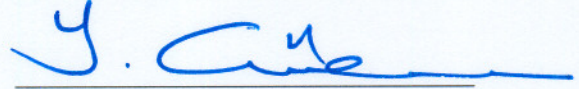
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2006

Title of the Thesis : **An Application of The Vehicle Routing Problem To A Glass
Manufacturing Firm**

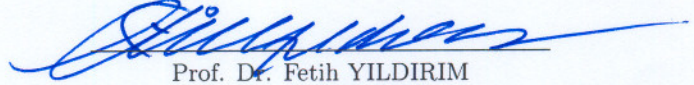
Submitted by **İpek Seyran**

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



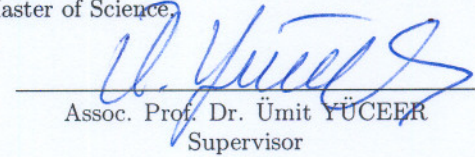
Prof. Dr. Yurdahan Güler
Director

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



Prof. Dr. Fetih YILDIRIM
Head of the Department

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



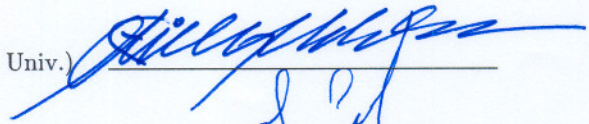
Assoc. Prof. Dr. Ümit YÜCEER
Supervisor

Examination Date: 12.08.2006

Examining Committee Members

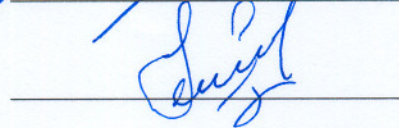
Prof. Dr. Fetih YILDIRIM

(Çankaya Univ.)



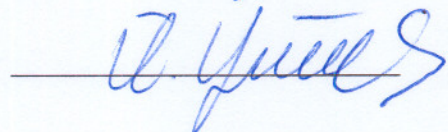
Prof. Dr. Serpil EROL

(Gazi Univ.)



Assoc. Prof. Dr. Ümit YÜCEER

(Çankaya Univ.)



STATEMENT OF NON-PLAGIARISM

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

Name, Last Name : İPEK SEYRAN

Signature : 

Date : 12.09.2006

ABSTRACT

AN APPLICATION OF THE VEHICLE ROUTING PROBLEM TO A GLASS MANUFACTURING FIRM

Seyran, İpek

M.S.c., Department of Industrial Engineering

Supervisor: Assoc. Prof. Dr. Ümit Yüceer

September 2006, 138 pages

This thesis presents an exact algorithm and a heuristic method for the delivery and transportation of glass plates for a glass manufacturing firm. A variant of the Capacitated Vehicle Routing Problem (CVRP) is proposed as a first attempt to solve the problem which minimizes total travelling of all the vehicles. Since the CVRP is known to be \mathcal{NP} -hard, the solution method cannot obtain a solution to the model. Therefore an exact algorithm which is a kind of set-covering-based algorithm is proposed next. The CVRP is modelled as a set covering (SC) problem. Then column generation (CG) method is applied to the linear relaxation of the SC problem. The branch-and-price algorithm is utilized in finding an integer solution on the solution of the CG procedure. Numerical experimentations reveals that exact algorithm is slower, and fails finding a solution to larger size problems. Consequently a heuristic is developed as a generalization of petal algorithm. Initialization of this algorithm

requires using some Travelling Salesman Problem (TSP) construction heuristics for finding a TSP tour, and a TSP improvement heuristic further improves the TSP tour. Then Petal Algorithm is applied to find all of the feasible petal routes to the TSP tour obtained. SP model helps the petal routes to find the best VRP routes. When the best VRP route is found, a VRP improvement heuristic attempts improving the VRP route. Finally, the number of delivery vehicles required and the vehicle routes are determined for the glass manufacturing firm.

Keywords: Capacitated Vehicle Routing Problem, Set Covering/Partitioning Problem, Column Generation Algorithm, Branch-and Price Algorithm, Generalized Petal Algorithm.

ÖZ

BİR CAM İMALAT FİRMASI İÇİN ARAÇ ROTALAMA PROBLEMİ

UYGULAMASI

Seyran, İpek

Yüksek Lisans, Endüstri Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ümit Yüceer

Eylül 2006, 138 sayfa

Bu tezde bir cam üretim firmasının dağıtım ve taşıma problemini çözmek için bir gerçek algoritma ve bir sezgisel yöntem geliştirilmiştir. İlk olarak yapılan yolu azaltmak amacıyla Kapasiteli Araç Rotalama problemi (KARP) olarak modellenen bir model kurulmuştur. KARP \mathcal{NP} -zor olarak bilinmektedir, bu nedenle kurulan model çözülememektedir. Bu yüzden bir tür küme kaplama temelli gerçek bir algoritma geliştirilmiştir. Bu algoritma için KARP, bir küme kaplama problemi olarak modellenmiştir. Daha sonra sütun üretme methodu küme kaplama probleminin doğrusal gevşemesine uygulanmıştır. Bir tam sayılı çözüm bulabilmek için Dalandır -ve- Fiyatlandır yaklaşımı uygulanmıştır. Gerçek algoritmasının yavaş çalıştığı ve büyük problemler için sonuç almanın zor olduğu görülmüştür. Bu nedenle petal algoritması geliştirilmiştir. Başlangıçta Gezgin Satıcı Problemi (GSP) yapım sezgisel yöntemleri kullanılarak bir GSP turu bulunmuştur ve GSP geliştirme sezgisel yöntemleri kul-

lanılarak geliştirilmiştir. Petal algoritması uygulanmıştır. Küme bölüntüleme modeli en iyi ARP rotasını bulmuştur. ARP rotasını geliştirmek için ARP geliştirme sezgisel yöntemleri uygulanmıştır. Bütün bunların sonucunda cam üretim firmasının araçlarının yaptığı yol miktarı, kullanılan araç sayısı ve araçların rotaları belirlenmiştir.

Anahtar Kelimeler: Kapasiteli Araç Rotalama Problemi, Küme Kaplama / Bölüntüleme Problemi, Sütun Üretme Algoritması, Dallandır -ve- Fiyatlandır Yaklaşımı, Genelleştirilmiş Petal Algoritması.

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor Assoc. Prof. Dr. Ümit YÜCEER for his support, guidance and especially for his patience throughout my thesis.

I am thankful to my chairman Prof. Dr. Fetih YILDIRIM, for his tolerance and support for my graduate studies.

I would also like to thank Assoc. Prof. Dr. Mustafa Köksal, Benhür SATIR, and my friends Ender YILDIRIM, İlter ÖNDER for being patient while replying my questions that are urgent for my study. Also thanks to my colleague Miray Hanım ASLAN.

I present my deepest thanks to my mother and father, İnci and Oğuz SEYRAN for supporting me and always making me feel comfortable.

Last but certainly not least, a very, very special person in my life, Engin TOPAN, and his family for their support during my study. His existence and compassion created an amazing support to me and my thesis.

Thanks to everyone I could not mention here.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM	iii
ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
CHAPTERS:	
1 Introduction	1
2 Related Studies	5
2.1 The Vehicle Routing Problem	5
2.2 A Specific Case of VRP: The Travelling Salesman Problem . .	11
2.3 Heuristics for VRP	14
2.3.1 Construction Heuristics for the VRP	14
2.3.2 Two Phase Methods for the VRP	15
2.3.3 Improvement Heuristics for the VRP	17
2.4 Exact Methods for the Vehicle Routing Problem	18
2.4.1 Branch and Bound Algorithms for VRP	19
2.4.2 Branch and Cut Algorithms for VRP	19
2.4.3 Set Covering Based Algorithms for VRP	20
3 Motivation and Problem Definition	22
3.1 The Specifications of the Problem	22
3.2 Glass Plates Arranging Algorithm	28
4 A Model for the Glass Transportation Problem	32
5 Solution Methods	38
5.1 An Exact Algorithm for the Problem	38
5.2 A Heuristic Approach for the Problem	46
5.2.1 The Generalized Petal Heuristics	46
5.2.2 The Improvement Heuristic for VRP	51

6	Numerical Experimentation	55
7	Conclusion	61
	REFERENCES	R1
	APPENDICES	A1
A	The Source Code Of Arranging The Glass Plates Algorithm	A1
B	The Gams Model Of The Capacitated Vehicle Routing Problem (CVRP) .	A6
C	The Source Code Of Exact Algorithm	A10
D	The Gams Model Of The Set Covering Problem For The Exact Algorithm.	A24
E	The Source Code Of Generalized Petal Heuristic	A26
F	The Gams Model Of The Set Covering Problem For The Generalized Petal Heuristic	A65
G	The Solution Of The Arranging Algorithm For A Given Data	A67
H	The Distance Matrix For The Given Data	A69

LIST OF TABLES

3.1	Capacity of the vehicles	27
3.2	An Example Of A Demand Of A Customer	31
6.1	Comparison Of The Exact Algorithm With The Optimum	56
6.2	Comparison Of The Gen.Petal Algorithm With The Optimum Solution	57
6.3	Best Solution Comparison Of The Gen.Petal With Other Heuristics	58
6.4	CPU Time Comparisons Of The Gen.Petal With The Other Heuristics	58

LIST OF FIGURES

2.1	An Example of A VRP	6
3.1	An Example Of An Order Of A Customer	24
3.2	The Backside And Side View Of A Truck	25
3.3	The View Of A Tripod	26
3.4	A View Of Truck When Loading	27
3.5	The Front View Of The Ordered Glasses In Table 3.2	31
5.1	The Flow Chart Of The Set-Covering-Based Exact Algorithm	45
5.2	2-Opt	49
5.3	String Exchange	52
5.4	The Flow Chart Of The Generalized Petal Heuristic	54

CHAPTER 1

Introduction

This thesis presents a model and solution methods for the delivery and a transportation problem of a glass manufacturing firm. This problem is modeled as an integer program. Subsequently, an exact algorithm and a heuristic is developed after investigating its mathematical properties.

This problem is summarized as follows: The glass plates will be delivered from the depot to the customers on time. For the delivery, the glass plates are to be loaded into the vehicles. This loading operation is very difficult because of the brittleness structure of the glass. The plates of glasses must be loaded carefully to the vehicles, otherwise the glasses may be broken or shattered during transportation. In this operation plates of glasses are arranged in some formulation according to a set of sequencing rules. The important thing is that if the glass plates are ordered properly than a lot of customers' orders can be loaded to a vehicle without exceeding the available capacity. There are four dimensions in defining the capacity of a vehicle. These are the length, the width, the height of the vehicle, and the weight capacity (hauling capacity) of the vehicle. After a truck is loaded, this truck must visit every customer only once and deliver the plates, and eventually return to the depot.

The problem is modeled as a Vehicle Routing Problem (VRP), more specifically as

a Capacitated Vehicle Routing Problem (CVRP). The structure of VRP, the types of VRP models, the solution methods and the parts of the solution methods are described in Chapter 2. Also the specific case of VRP which is known as Travelling Salesman Problem (TSP), and the solution methods are explained.

In Section 3.1 the description of the delivery and transportation problem is discussed to explain all the dimensions and the difficulty of the problem.

As mentioned in Section 3.1 the plates of glasses should be put in an arrangement properly before being loaded to the vehicle. This ordering operation is done heuristically by the workers responsible from the delivery and transportation. An algorithm is developed in Section 3.2 to arrange the glass plates which takes the experience and the instructions of the workers. This is a very valuable part of this research because the four dimensions of the glass plates can be obtained by using the *Sequencing the Customers' Orders Algorithm*. The output of this algorithm is used as an input to the proposed model. The output of Sequencing the Customers' Orders Algorithm is an input for the parameters of the model (CVRP).

After obtaining the problem parameters in Chapter 4, an integer program model is constructed to solve the problem. This model is a kind of VRP known as CVRP. A four dimensioned and a one dimensioned models are derived. It is known in literature that VRP is \mathcal{NP} -hard. Therefore the solution of the model cannot be obtained for larger problems. For this reason, alternative solution methods are investigated.

In Chapter 5 the alternative solution methods for the problem are developed. An exact algorithm is used in Section 5.1. This approach is a set-covering-based algorithm. In

this algorithm VRP is modeled as a set covering problem and it is assumed that the distances between three locations satisfies triangular inequality. By using the linear relaxation of the set covering problem and the dual of this relaxation problem, a lower bound is obtained for the set covering problem via the column generation method. Dual of the relaxation of the set covering problem is used to check the optimality of the relaxation problem. After the column generation procedure, the branch-and-price algorithm attempts finding an integer solution. The exact algorithm is not fast enough to solve the model and fails for larger size problems. Thus some other methods are needed.

One alternative method is developed for the problem and explained in Section 5.2. This is a heuristic method based on the generalized petal algorithm. Four construction TSP heuristics are used to determine a TSP tour. After finding a TSP tour, then an improvement TSP heuristic is applied to the found TSP tour. Then generalized petal algorithm is used to find all of the feasible petal routes. By the help of a set partitioning model the best VRP routes are selected from inside the feasible petal routes. Finally, a VRP improvement heuristic is applied to the solution of the set partitioning algorithm. After all the solution of the problem is found.

In Chapter 6 the numerical results of the solution methods are given. The proposed heuristic is tested on twelve benchmark problems described in Christofides, Mingozi and Toth (1979). After that the proposed model, the exact algorithm and the proposed heuristic is applied to the delivery and transportation problem of the firm. These solutions are compared with the daily data obtained from the firm. As the result of the constructed model the same number of trucks are obtained for the delivery

and the transportation problem. But the distance travelled is reduced by 98.3 km when compared to usual implementation of the methods of the firm.

CHAPTER 2

Related Studies

In this Chapter to understand the structure of the Vehicle Routing problem (VRP) a brief explanation is described. The constructed models and the solution methods of VRP in the literature are considered.

2.1 The Vehicle Routing Problem

The distribution of goods to the customers is one of the most important logistics problem. One of the most popular problem on this subject is the Vehicle Routing Problem (VRP). There are various applications of this problem; school bus routing, inventory routing, pickup and delivery, and some scheduling problems. Therefore there are various names, definitions, models for VRP. Most common names used in the literature for the problem are VRP, pickup and delivery problems, vehicle scheduling and truck dispatching problems. The problem may differ in terms of number of depots that the goods stored, vehicle fleet types, objective, capacity requirements of the vehicles (weight, volume, length, height), inclusion of time concept. Even more generalized version such as pickup and delivery problems, inventory routing problems can be accepted as a part of VRP. Therefore a common definition for the problem is difficult because of the variety in terms of boundaries of the problem from one

case to another. But in general VRP is concerned with delivering goods from one or more depots to a number of customers or cities by means of a fleet of vehicles. Thus VRP involves the determination of a set of routes for each vehicle where each vehicle starts from and returns to the depot (Figure 2.1). The objective is obviously to minimize the total transportation cost (distance). Each vehicle has a capacity, and each customer has a demand that uses some portion of the vehicle capacity. The problem with identical vehicle capacities is known as Capacitated Vehicle Routing Problem (CVRP). Otherwise it is called heterogeneous fix fleet vehicle routing problem (HVRP). In this thesis, the term "the cost" and "the distance" are equivalent and used interchangeably.

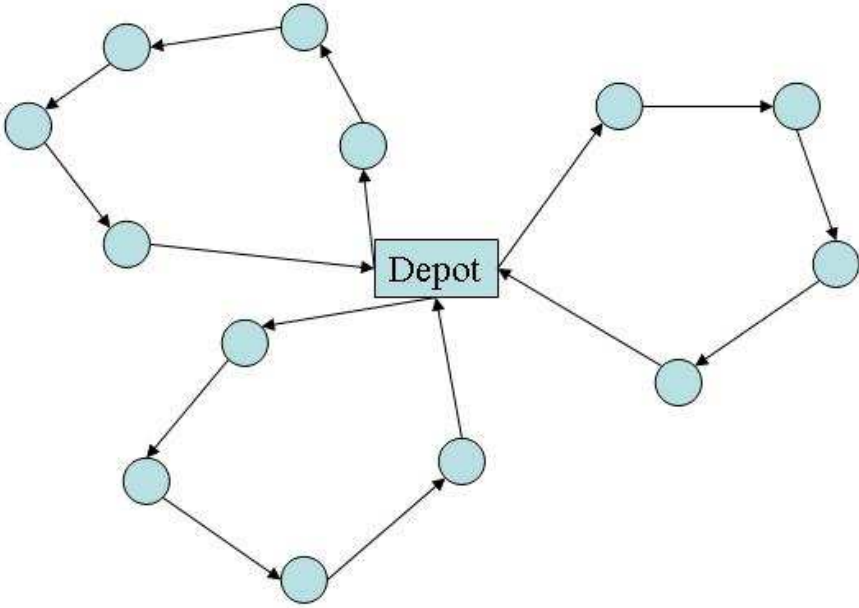


Figure 2.1: An Example of A VRP

There are various VRP models in the literature as summarized by Toth and Vigo (2002). Some notations are provided in explaining the most common models in the literature.

Let $G = (N, E)$ be a complete graph where $N = \{0, \dots, n\}$ is the set of nodes and E is the set of edges for the graph G . Vertices $i = 1, \dots, n$ are the customers whereas vertex 0 represents the depot. d_{ij} (c_{ij}) is the transportation cost corresponding to the edge $(i, j) \in E$. If $c_{ij} = c_{ji}$ for every (i, j) , then the problem is referred as symmetric VRP (otherwise asymmetric). In this thesis Euclidean distances are used. It is assumed that all the costs (distances) are symmetric. Also in most cases, the triangular inequality is satisfied by any three elements of the set N .

$$c_{ik} + c_{kj} \geq c_{ij}, \quad \forall i, j, k \in N \quad (2.1)$$

Another important aspect of VRP is the vehicle types. If the vehicles are identical, the problem is known as Capacitated Vehicle Routing Problem (CVRP). If vehicles are not identical the problem is known as Heterogeneous Fleet Vehicle Routing Problem (HFVRP).

One integer programming representation of the problem is the two index vehicle flow formulation constructed for CVRP (Toth and Vigo (2002)). It uses n^2 binary variables. The decision variables for the model are given next.

$$x_{ij} = \begin{cases} 1 & \text{if edge } i, j \in E \text{ belongs to the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

Then the mathematical model is given by;

$$\min \quad \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij} \quad (2.2)$$

Subject to

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in C \quad (2.3)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in C \quad (2.4)$$

$$\sum_{i \in N} x_{io} \leq m \quad (2.5)$$

$$\sum_{j \in N} x_{0j} \leq m \quad (2.6)$$

$$\sum_{i \notin S} \sum_{i \in S} x_{ij} \geq r(S) \quad \forall S \subseteq C, S \neq \emptyset \quad (2.7)$$

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

where m is the fleet size, and $C = N \setminus \{0\}$. Constraints (2.3) to (2.6) are known as the degree constraints. The Constraints (2.3) and (2.4) guarantee that every customer is visited exactly once (one for entering and one for leaving each customer). Constraints 2.5 and 2.6 imposes a bound on the number of arrivals and departures for the depot. Naturally this bound is the fleet size for the customers and for the depot. Constraint (2.7) is known as the capacity cut constraints and guarantees the connectivity (subtour elimination) of the solution and the vehicle capacity requirements where $r(S)$ is the minimum number of vehicles required for the set S . In this research, Bin Packing problem solution is preferred as a lower bound instead of $r(S)$.

Alternatively, capacity cut constraints can be replaced by generalized subtour elimination constraints which is inspired from Travelling Salesman Problem (TSP) as in the Constraint (2.8).

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - r(S) \quad \forall S \subseteq C, S \neq \emptyset \quad (2.8)$$

The number of constraints in both of the capacity cut constraints and the generalized subtour elimination constraints grow exponentially with parameter n . Therefore the separation procedure is utilized in most cases. In this procedure the problem starts with some of these constraints, then if a violation is observed, the violated constraint is included into the model. These separation problems are also difficult. There are both exact and heuristic separation algorithms which requires identifying the violated constraints and then adding these constraints to the previous ones. These type of procedures are accelerated within a branch and cut (and price) procedure, Fukasawa et al. (2006), Ralphs et al. (2003), Blasum and Hochstattter (2002), Lysgaard et al. (2004), Achuthan et al. (2003), Vigo and Toth (2002).

A third approach is to use Miller, Tucker and Zemlin (MTZ) (1964) constraints (Constraints 2.9 and 2.10) instead of the generalized subtour elimination constraints. Although with the use of MTZ constraints, Linear Relaxation of CVRP is much weaker than both the capacity cut constraints and the generalized subtour elimination constraints. But the advantage of using MTZ constraints is that the number of such constraints is polynomial.

$$u_i - u_j + CLx_{ij} \leq CL - L_i \quad \forall i, j \in C, i \neq j, \text{ such that} \quad (2.9)$$

$$d_i + d_j \leq CL$$

$$D_i \leq u_i \leq C \quad \forall i \in C \quad (2.10)$$

The two index vehicle flow problem does not help if one needs the information about which vehicle is assigned to which route. This is the case when there is more than one vehicle type (e.g. HVRP). In such cases three-index formulation with n^2K variables is used. The following model (Equations (2.11) to (2.16)) is a well known three-index representation of the routing problems in the literature.

$$y_{ik} = \begin{cases} 1 & \text{if customer } i \text{ is visited by the vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

,

$$x_{ijk} = \begin{cases} 1 & \text{if customer } j \text{ is immediately visited} \\ & \text{after customer } i \text{ by the vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

Then the mathematical model is given as follows.

$$\min \quad \sum_{i \in N} \sum_{j \in N} d_{ij} \sum_{k \in K} x_{ijk} \quad (2.11)$$

Subject to

$$\sum_{i \in N} D_i y_{ik} \leq C_k \quad k \in K \quad (2.12)$$

$$\sum_{k \in K} y_{ik} = 1 \quad \forall i \in C \quad (2.13)$$

$$\sum_{k \in K} y_{0k} = m \quad (2.14)$$

$$\sum_{j \in N} x_{ijk} = \sum_{j \in N} x_{jik} = y_{ik} \quad \forall i \in N, \forall k \in K \quad (2.15)$$

$$\sum_{i \notin S} \sum_{i \in S} x_{ij} \geq y_{hk} \quad \forall S \subseteq C, h \in S, k \in K \quad (2.16)$$

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

$$y_{ik} \in \{0, 1\} \quad \forall i \in N, k \in K$$

In addition to these, VRP problems can be formulated as a Set Partitioning problem (SP) which is proposed by Balinski and Quandt, (1964). The advantage of formulating VRP as set partitioning is this formulation is very tight. However it uses exponential number of binary variables. The notation for the set partitioning problem is follows;

$$\alpha_{ir} = \begin{cases} 1 & \text{if customer } i \text{ is served in route } r \\ 0 & \text{otherwise} \end{cases}$$

and the decision variable is;

$$x_r = \begin{cases} 1 & \text{if route } r \text{ is in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

Then the mathematical formulation is then as follows.

$$\min \quad \sum_{r \in R} d_r x_r \quad (2.17)$$

Subject to

$$\sum_{r \in R} \alpha_{ir} x_r = 1 \quad \forall i \in C \quad (2.18)$$

$$x_r \in \{0, 1\} \quad \forall r \in R$$

where R is the set of all feasible routes.

2.2 A Specific Case of VRP: The Travelling Salesman Problem

The VRP is a combination of routing and packing problem. Both the travelling salesman problem (TSP) and the bin packing problem (BPP) are special cases of the VRP. To be more specific, TSP is equivalent to the VRP with $m = 1$ and $D_i = 0$, whereas BBP is equivalent to the VRP when edge costs (distances) equals to 0. Since BBP and TSP are \mathcal{NP} -hard combinatorial problems, VRP is also \mathcal{NP} -hard.

In this study we also refer to the TSP literature due to the similarities of VRP and TSP. To be more specific, we use well known TSP heuristics in our solution method.

Therefore in this subsection also a brief introduction to TSP and a summary of well known TSP heuristic in the literature will be given.

Given a graph $G = (N, E)$ a tour that traverses each node exactly once is called a Hamiltonian tour. TSP can be stated as finding the Hamiltonian tour in which the minimum distance travelled. There are a lot studies for both exact and heuristic solution methods proposed in the literature. Besides the sophisticated metaheuristics, there are very known simple classical heuristics that is tailored for TSP. Some of them are known as TSP *construction heuristics* which starts with an incomplete solution and implement a specific procedure until a complete solution is obtained. The most common ones are the following ones:

Nearest Neighborhood Heuristics: This is the simplest and the most common TSP heuristics. The idea is to include the nearest node (customers) to the last included node at each step.

Insertion Heuristics: It starts with a small tour, and then the tour is extended by inserting the remaining nodes by using an insertion rule. According to the insertion rule it has different versions such as (nearest insertion, farthest insertion, cheapest insertion, random insertion etc.).

Savings Heuristics: It is also known as Clark and Wright (1964) Heuristics. Starting with n two-node tours connected to the depot, at each iteration the subtours with the highest savings are merged until a Hamiltonian tour is obtained.

Greedy Heuristics: It is an approximated version of savings. At each step the

shortest edge is selected and added to the tour until a Hamiltonian tour is obtained.

Another type of TSP heuristics is the *improvement heuristics*. It starts with a complete solution, and then at each step the solution is improved using some heuristic rules until there is no possible way to improve the solution. The most common ones are as follows:

Node Insertion: Starting with an initial tour, a node is removed and reinserted to the best possible location.

Edge Insertion: Starting with an initial tour, an edge is removed and reinserted to the best possible location.

2-Opt Exchange: Starting with an initial tour, two edges are eliminated and they are reconnected in the other way.

k-Opt Exchange: Starting with an initial tour, k edges are eliminated and they are reconnected in the best possible way out of other $2^k - 1$ ways.

Lin-Kernighan Type Heuristics: is a variable k-opt algorithm. At each step the most suitable k value is selected. Although the algorithm is a complicated one, it is one of the most powerful tools to solve TSP in the literature (Lin and Kernighan (1973) for further details).

2.3 Heuristics for VRP

The exact solution for the CVRP is only reasonable for small size problems. As the size of the problem size increases the only way to deal with such problems is to implement heuristics procedures. Therefore there are classical heuristics and metaheuristics proposed for the problem. In this study we are concentrated on the classical heuristics and in this subsection a brief summary of those heuristics is given. Although there are different ways to classify these classical heuristics, a common classification is proposed by Laporte and Semet in Toth and Vigo (2002). Accordingly, classical heuristics are classified in three groups; *construction heuristics*, *two-phase methods*, *improvement heuristics*.

2.3.1 Construction Heuristics for the VRP

Similar to the construction heuristics in TSP, VRP construction heuristics involves *savings and insertion heuristics*. There are two well known savings algorithms in the literature as follows:

Clarke and Wright Savings Algorithm: (Clarke and Wright (1964)) The algorithm requires computation of saving for each possible merge using the following formula:

$$s_{ij} = d_{i0} + d_{0j} - d_{ij} \quad \forall i, j \in C \quad (2.19)$$

At each after step savings are calculated and sorted in nonincreasing order. Starting from the top the feasibility of each merge is checked one by one and if the corresponding merge is feasible, it is immediately performed. This procedure is continued until

there is no improvement.

Matching Based Savings Algorithms: There are three well known matching based savings algorithms in the literature (Desrochers and Verhoog (1989), Altinkemer and Gavish (1991), Wark and Holt (1994)). The main idea is to solve a max-weight matching problem in which weights are the savings obtained by merging of routes. In this problem maximum weight subset of edges are selected such that for every node, at most one edge in that subset includes the node as an end point.

Sequential Insertion Heuristics: These algorithms are classified in two groups such that parallel insertion and sequential insertion. The most common ones are proposed by Mole and Jameson (1976) and Christofides, Mingozzi and Toth (1979). The first one has only a sequential version whereas the latter includes the both versions. In the parallel version at each step an unassigned customer, insertion cost of the customer to all routes is evaluated. In the sequential case at each step only the insertion cost of the customer to the current route is evaluated. According to the insertion costs the unassigned customer with the minimum insertion cost is added to the corresponding route.

2.3.2 Two Phase Methods for the VRP

As it is mentioned before VRP is a combination of routing and packing problem. Therefore some researchers propose different heuristics that solves these two subproblems sequentially. So there are are mainly two categories of heuristics in two-phase methods. One is the *cluster first, route second* whereas the second one is the opposite,

route first, cluster second. The most common cluster first, route second methods are as follows:

Sweep Algorithm: In the order due to rotating a ray around the depot the clusters are obtained. Then for each cluster a TSP is solved to obtain the optimal routing of the corresponding cluster (Gillet and Miller (1974)).

Fisher and Jaikumar Algorithm: After selecting seed customers insertion cost of each customer to each cluster is obtained. Using these costs in a generalized assignment problem clusters are obtained. Finally a TSP is solved to obtain the optimal routing of the corresponding cluster (Fisher and Jaikumar (1981)).

Bramel and Simchi-Levi Algorithm: Seed customers are determined by solving a capacitated location problem, and then the uncovered customers are included into the clusters centered at the seed customers according to least insertion cost rule (Bramel and Simchi-Levi (1995)).

Petal Algorithms: It is an extension for the sweep algorithm. Here the routes which are obtained in the order due to rotating a ray around the depot are called as petals. Finally out of many petals optimal petal selection is achieved by solving the set partitioning problem proposed by Balinski and Quandt (1964).

$$\min \sum_{p \in S} d_p x_p \quad (2.20)$$

Subject to

$$\sum_{p \in S} \alpha_{ip} x_p = 1 \quad \forall i \in C \quad (2.21)$$

$$x_p \in \{0, 1\} \quad \forall p \in S$$

where S is the set of all feasible petals, $x_p = 1$ if and only if route k belongs to the solution, α_{ip} is the binary parameter which equals to 1 if customer i belongs to petal p and d_p is the cost of petal p .

Later some other variants of the petal generation scheme are proposed. They propose to order the customers in different fashions. For example in the sweep algorithm customers are ordered in the radial order, but with the help of these variants any other orderings are also possible. The key point behind the petal algorithms is the following: If all the routes, that is the set S , are obtained according to an ordering rule, then the resulting set partitioning problem becomes totally unimodular. In that case solving set partitioning problem as an LP gives an IP solution. The routes that are obtained according to other possible rules are called as generalized petals. In this we utilize generalized petals in the heuristic method proposed (Ryan et al. (1993) and Renaud et al. (1996) for further details).

There are only a few studies published on the *route first, cluster second* methods. These algorithms start with constructing a single giant TSP tour ignoring the capacity limitation of a truck. Later this tour is decomposed into feasible tours. Consequently the problem becomes a shortest path problem (Beasley (1983)).

2.3.3 Improvement Heuristics for the VRP

There are two versions of the improvement heuristics for the VRP. In one of them each route is improved independent of the other routes, whereas in the second one the improvement procedure is applied simultaneously applied to all routes.

Single Route Improvements: Because the procedure is applied to a single route which is, in fact, a TSP, the TSP heuristics applied to the corresponding route improvement. These heuristics are discussed in Subsection 2.2, multiroute improvements are discussed separately next.

Multiroute Improvements: There are many multiroute improvement procedures in the literature. But the most common ones are classified by Van Breedam (1994). A string definition will be useful for explaining this classification. A string is a chain of consecutive nodes (e.g. a string of single node is the node itself, whereas a string of two nodes corresponds to the two nodes and the edge between these nodes). Accordingly there are three main variants of multiroute improvement heuristics; *string cross*, *string exchange*, and *string relocation*. In the string cross, two strings of different routes are exchanged by crossing the endpoints of the corresponding strings, whereas in the string exchange two strings are exchanged between two routes. In the string relocation a string is removed from the route it belongs to and attached to another route.

2.4 Exact Methods for the Vehicle Routing Problem

Because there are so many exact algorithms proposed for VRP, it is difficult to classify and summarize these algorithms. Although there are alternative classifications, the one proposed by Toth and Vigo (2002) is preferred in this review (Laporte (1992) gives an alternative classification). Thus these algorithms can be grouped in three categories as; branch and bound (B & B), branch and cut (B & C) algorithms, and the set covering based exact methods.

2.4.1 Branch and Bound Algorithms for VRP

There are different exact B & B algorithms for VRP. These algorithms differ within themselves in using the lower bounding procedures. In some of these B & B algorithms, the lower bound are obtained by dropping the capacity cut constraints (Equation (2.7)) or generalized subtour elimination constraints (Equation (2.8)). The resulting problem is an assignment problem or a b-matching problem, with the loss of connectivity and capacity requirements in the model. For example Laporte et al. (1986) proposed a B & B algorithm in which the relaxation for each branch are obtained by dropping capacity cut constraints (Equation (2.7)) in the model with Equations (2.2) to (2.7). In an alternative relaxation scheme instead of dropping the corresponding constraints they are weakened. But at this time outdegree constraints (Equation (2.4)) are dropped out (Christofides, Mingozzi and Toth (1981) and Fisher (1994) for this alternative scheme). There are also more sophisticated lower bounding procedures in the literature based on solving lagrangean relaxation (Fisher (1994)), dual of the relaxation to set partitioning formulation (Hadjiconstantinou, Christofides and Mingozzi (1995) and additive bounding procedures (Fischetti et al. (1994)).

2.4.2 Branch and Cut Algorithms for VRP

B & B algorithms are reasonable only if the number of constraints is in tolerable levels. When the number of constraints is large or in cases where valid inequalities are known, a different procedure which is known as cutting plane branch and cut (B & C) algorithm may be very beneficial. At each branch the algorithm starts with solving

the LP relaxation of the problem. If the solution to this problem gives IP solution, then this solution is an optimal solution for the IP problem of the overall problem. If not, cuts (valid inequalities) that are violated are added to the model (as long as these violated cuts are detected). If the procedure is not terminated with an optimal IP solution, then branching is performed. The main problem is how to detect those cuts at each iteration. For this purpose a scheme which is known as separation procedure is used. Many of the cuts (valid inequalities) of CVRP are well defined and known in the literature (e.g. capacity cuts, generalized capacity cuts, framed capacity cuts, valid inequalities for TSP, valid inequalities for BBP etc..). There are both exact and heuristics separation algorithms to generate the inequalities that are violated (Blasum and Hochstattler (2002), Ralphs et al. (2003), Achuthan et al. (2003) and Lysgaard et al. (2004)). A more sophisticated variant of B & C algorithm is the branch and cut and price (B & C & P) algorithm in which B & C and pricing (column generation) techniques are combined. B & C & P algorithms are known to be the most powerful exact algorithms in the literature (Fukasawa et al. (2006)).

2.4.3 Set Covering Based Algorithms for VRP

As it is mentioned before (Constraints 2.17 and 2.18) VRP can be modeled as a set covering/partitioning problem (only difference is due to equality becomes an inequality). In order this set covering model to be valid, the set of all routes, R , should be complete. However the number of routes grows exponentially. In set covering based algorithms the idea is to solve the LP relaxation of the set covering model without enumerating all the routes. Later, the routes that are profitable are generated us-

ing a column generation scheme. When this procedure is embedded into a branch and bound algorithm the method is known as branch and price. Although the idea seems to be simple, solving the corresponding column generation problem is difficult. Agarwal et al. (1989) proposed a branch and bound algorithm, whereas Bixby et al. (1997) developed a cutting plane procedure to solve this column generation problem. Desrochers et al. (1992) used dynamic programming to find a lower bound for the solution of the column generation problem rather than solving it. In this thesis we use dynamic programming to solve the column generation problem for our set covering based exact algorithm.

CHAPTER 3

Motivation and Problem Definition

In order to understand the delivery and the transportation problem, the definitions and the specifications of the problem are given in this chapter. Also a proposed algorithm is described in the previous sections which is used to arrange the glass plates in an order.

3.1 The Specifications of the Problem

An intermediate glass firm purchases its raw material from a glass manufacturer firm in standard plates of dimensions. The glass is cut according to the needs of each individual customer in appropriate dimensions and treated for obtaining heat treated glass, tempered glass, float glass, float mirror, parapet and faade cladding glass, the special quality home glasses, the decorative glasses, bricks and parquets made of glass, glass doors, the security glasses of tempered and counter-assault units.

The firm has a potential of 1000 to 1500 customers for which they cut and treat glass in a year. Occasionally the number of customers goes up to 5000 in a year. Most of the customers are located in the vicinity of Ankara. The firm has also some customers in other cities in Turkey and additionally abroad. The working policy of

the firm is to receive orders first, then do the cutting and treating the glass according to the specifications of the customers. The customers orders contain different sizes and various treatments of the glass. Consequently, there are various sizes and types of glass to be distributed and delivered.

The cutting problem is solved by using an optimization package specially developed for glass industry. This is a form of cutting stock problem as known in the literature. The distribution and timely delivery of the glass cut and treated needs further research and analysis. The main thrust of this research and thesis is to tackle the distribution and the delivery of the glass plates to the customers.

This problem has two main parts. One part is to load a vehicle for transport and delivery. The customers' orders vary in sizes of length, width and height. This variety of the size of the glass causes some problems in loading the vehicle for delivery to a group of customers. The brittleness structure of the glass, makes the sequence of loading the glasses to the vehicles very important. The stress between the plates of the glass should be minimized in order to prevent the glass broken or shattered during the transportation. Subsequently the arrangement of the customer' orders to be loaded to a truck creates a loading problem. The Figure 3.1 displays three of a customer's orders. Different columns are used to separate a customer's orders from the other customers' orders. In other words each customer's order is arranged as a column in the truck and the other customers' orders are put in separate columns. The glasses shown in Figure 3.1 are not put in order. If they are loaded to the truck as shown, there is a risk of glass plates shattered or broken. Thus the plates of the glass should be put in an order according to their dimensions. The systematic way of

placing the plates of the glass will be explained in Section 3.2.



Figure 3.1: An Example Of An Order Of A Customer

They put the plates of the glass in some order when they load to a truck. The order of loading the customers' orders to the truck affects the routing of the truck during the delivery and transportation. By the way at same time only two vehicles can be loaded because of the location of the factory. After completing the loading of those two trucks, more trucks can be loaded (if needed).

Each vehicle has a hauling capacity. Thus the capacity of the vehicle must be a serious restriction in loading the glass plates. The capacity of a truck can be expressed in four different dimensions: the length, the height, the width, and the mass of the glass. In addition, there is a safety factor of hauling glass by the state. The Figure 3.2 shows the backside and side views of the truck for loading glass plates the length, height and width. Consequently loading cargo to a vehicle requires considering all of the four

capacity dimensions. The length of the total glass plates loaded cannot exceed the length, their maximum height cannot exceed the height of the truck and the width of the glass plates put on top of each other cannot exceed the width the truck of the truck. In addition, the total weight of the glass plates loaded cannot exceed the hauling capacity of the truck. The firm owns six vehicles and additional vehicles can be rented for delivering glass plates to the customers if needed.

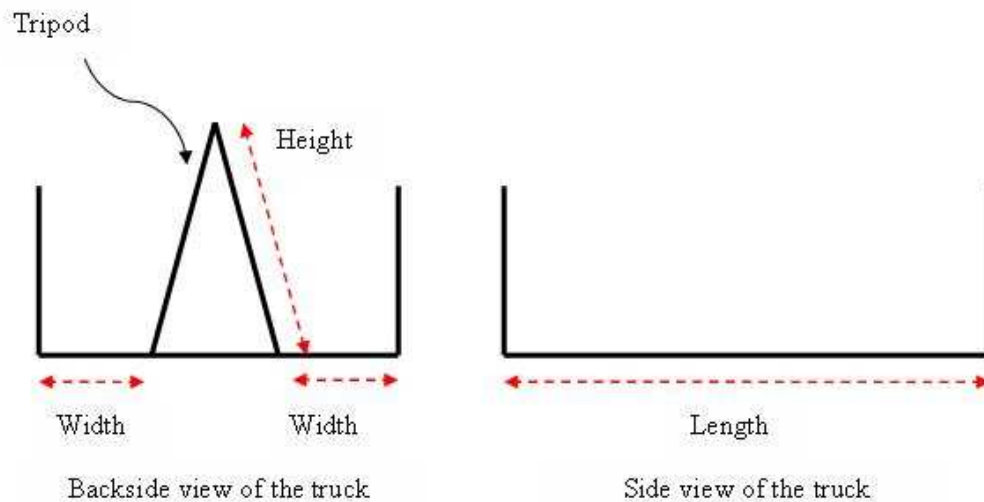


Figure 3.2: The Backside And Side View Of A Truck

The plates of glass are leaned onto one side of the tripod and then tied by a rope. Tripod is securely placed in the middle of the cargo deck of the truck. There is a ten cm safety space left between the columns of two customers' glass plates. A view of a tripod is shown in Figure 3.3.

The distance between the storage area and the loading dock is about two or three meters. A picture of the workers while they are loading the truck is provided in Figure 3.4. The firm has six trucks for the delivery and transportation of the customers.



Figure 3.3: The View Of A Tripod

Table 3.1 shows the capacities of these six vehicles. Even though there are two different types of vehicles, modelling and solving the loading problem, it is assumed that all the vehicles have identical features in terms.

An important aspect of transportation problem is to deliver the customers' orders on time. The aim of delivering customers' demands is to minimize the distance travelled and/or if possible to minimize the number of trucks used.

In literature this kind of distribution problems are called the *Capacitated Vehicle Routing Problem (CVRP)*. Each day they have to distribute lots of customers' demands. The loading portion of the problem is a *Bin Packing Problem (BPP)* and the routing part of the problem is known to be *Vehicle Routing Problem (VRP)* in the literature.

We need to know the length, the width, the height and the weight of each customer

Table 3.1: Capacity of the vehicles

	Capacity			
Trucks	Length (m)	Height (m)	Width (m)	Mass (tone)
1	6	2.5	0.7	8
2	6	2.5	0.7	8
3	6	2.5	0.7	8
4	6	2.5	0.7	8
5	6	2.5	0.7	8
6	5	2.3	0.6	6



Figure 3.4: A View Of Truck When Loading

in order to construct the model discussed in Chapter 4. The algorithm in Section 3.2 explains the way of knowing these four aspects values by the use of ordering the plates of glass algorithm.

3.2 Glass Plates Arranging Algorithm

Glass transportation differs from transporting the other goods because of the brittle-ness structure of the glass. That's why loading the plates of the glass to the vehicles is an important problem as mentioned in Section 3.1. The stress that each glass brings to the other plates is a static problem to be taken care of while ordering the plates of the glass onto each other.

This ordering glass plates problem is important for the model of Chapter 4. This forms the data for the length, height, width and the mass of the columns of the glass plates. In other words the output data of the glass ordering problem is going to be used as the input data of the proposed model. If the dimensions of the customer glass packages should be found, then the capacities of each customers can be determined.

The ordering of the glasses are completely/fully intuitional. The exact way of ordering glasses cannot be found because of the variety of the glass sizes. By the way while loading the glasses one should be careful for the type of the glasses. This means that heat treated glasses, tempered glasses and float glasses would be loaded separately. Because all of these types of glasses have different characteristics. The load and stress for different types of glasses varies from each other. In general, customers want only one type of glass such as heat treated glass or tempered glass or some other type, but sometimes they demand different types of glasses in a single order. Subsequently the workers try to put the glasses in order by looking without the type of the glass in an order of tempered glass first, heat treated glass second and float glass third. But they take measures/precautions for not to brake the glasses by putting special papers and

foams between the glasses. As observed by the management, the loss of glass during transportation is a lot less than the loss in production. The firms statistics tells us that in a year approximately five plates of glasses are broken during transportation.

The methods that the firm use for putting the order of the plates of the glasses while loading is based on sequencing the glasses according to the sizes. They load the largest glass first then the next largest glass. There is one important rule that at least one of the sides should be loaded one on the top of the other. The other rule is to load the plates of glasses vertically if the ratio of the height of the glass and the length of the glass does not exceed 1.75. If that ratio exceeds 1.75 then they load the plates of glasses horizontally. This ratio is determined intuitively by the experience of the workers. The plates of glasses longer than the tripod can be loaded easily by increasing the height of the tripod by assembling an extension at the top of the tripod. If the width of a customer's demand exceeds the truck's width, a new column would be opened up to the next of the column customer's glasses.

In our model, we take the workers' experience in to consideration on of height, length, width and mass. We assume the following instructions for the arranging of glass plates:

- Glasses are loaded vertically.
- Wider glass plates are loaded first.
- A longer glass plate with less width than some other plate, is loaded in front of the other one.
- Different columns are used for each customer.

- If the customers' demands exceed truck's width, then a new column is opened for the customer. Then both columns' length are added together in determining the length of each customer.

The algorithm is coded in Matlab and an Intel Centrino 1.7 processor computer is used for solving the algorithm. The code for this algorithm is given in Appendix A. An example of the glass plate arranging algorithm is shown below.

The customers in Table 3.2 demand a total of 10 glass plates all in different sizes. The above algorithm will put them in an order for loading. As a result the glasses in the Table 3.2 are sequenced as 5-1-2-6-3-7-8-4 (5th glass is in front of all of the glasses and 4th glass is behind all of the glasses), with a length of 1074 mm, width of 180 mm, height of 1241 mm and a total mass of 0.1218 tone. This means that the data needed for the model is determined as the length, height, width and mass of the customer. This data is used for constructing the capacity constraints of the model. In Figure 3.5 the front view of the ordering of the glasses are shown.

Table 3.2: An Example Of A Demand Of A Customer

Glasses	Dimensions			Mass(tonne)	Unit
	Length(mm)	Height(mm)	Width(mm)		
1	194	1241	18	0.004738	1
2	471	1111	18	0.010298	1
3	601	1091	18	0.012904	1
4	1074	1194	18	0.050474	2
5	198	609	18	0.002325	1
6	574	609	18	0.006879	1
7	704	609	18	0.008438	1
8	1074	609	18	0.025744	2

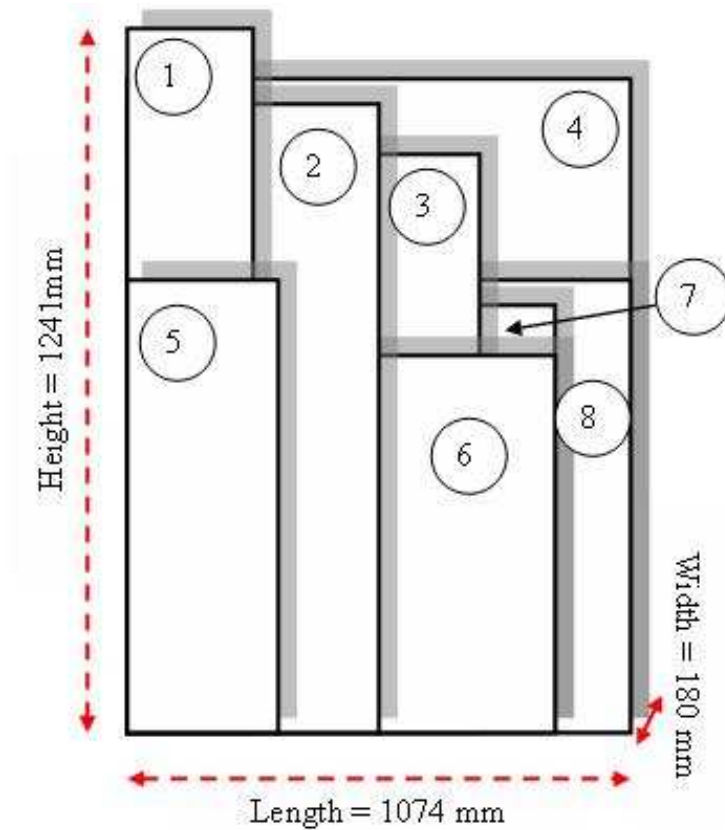


Figure 3.5: The Front View Of The Ordered Glasses In Table 3.2

CHAPTER 4

A Model for the Glass Transportation Problem

An integer linear programming model is developed to solve the problem described in the previous chapter. The model is a variation of the Capacitated Vehicle Routing Problem (CVRP).

Let n = is the number of customers, and $C = \{1, 2, 3, \dots, n\}$ represent the set of customers and the depot location by 0. Then $G = (N, E)$ be a complete directed graph representing the Vehicle Routing network where $N = C \cup \{0\} = \{0, 1, 2, 3, \dots, n\}$ is the set of nodes and $E = \{(i, j) : i, j \in N, i \neq j\}$ is the set of edges (arcs). The additional notations of the mathematical model are listed below;

m = number of delivery vehicles, and the index set $K = \{1, 2, \dots, m\}$ represents the m vehicles.

CW_k = the width of the vehicle $k \in K$,

CL_k = the length of the vehicle $k \in K$,

CH_k = the height of the vehicle $k \in K$,

CM_k = the hauling capacity in tonnes of the vehicle $k \in K$,

d_{ij} = distance from customer i to customer j for $(i, j) \in E$

W_i = the width of the plates of the customer $i \in C$,

L_i = the length of the plates of the customer $i \in C$,

H_i = the height of the plates of the customer $i \in C$,

M_i = the weight of the customer $i \in C$

The distance matrix is obtained from a map of Ankara, and it is symmetric. Thus we deal with a symmetric vehicle routing problem, and note that, $d_{ii} = \infty$ for all $i \in N$.

$$y_{ik} = \begin{cases} 1 & \text{if customer } i \text{ is visited by the vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

,

$$x_{ijk} = \begin{cases} 1 & \text{if customer } j \text{ is immediately visited} \\ & \text{after customer } i \text{ by the vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

and

u_{ik} = the load of the vehicle after visiting customer i by the vehicle k .

Then the mathematical model of the firm's problem is given by as follows.

$$\min \quad \sum_{i \in N} \sum_{j \in N} d_{ij} \sum_{k \in K} x_{ijk} \quad (4.1)$$

Subject to

$$W_i y_{ik} \leq CW_k \quad \forall i \in N, \forall k \in K \quad (4.2)$$

$$\sum_{i \in N} L_i y_{ik} \leq CL_k \quad \forall k \in K \quad (4.3)$$

$$\sum_{i \in N} M_i y_{ik} \leq CM_k \quad \forall k \in K \quad (4.4)$$

$$H_i y_{ik} \leq CH_k \quad \forall i \in N, \forall k \in K \quad (4.5)$$

$$\sum_{k \in K} y_{ik} = 1 \quad \forall i \in C \quad (4.6)$$

$$\sum_{k \in K} y_{0k} = m \quad (4.7)$$

$$\sum_{j \in N} x_{ijk} = \sum_{j \in N} x_{jik} = y_{ik} \quad \forall i \in N, \forall k \in K \quad (4.8)$$

$$u_{ik} - u_{jk} + CL_k * x_{ijk} \leq CL_k - L_i \quad \forall i, j \in C, i \neq j, \text{ such that} \quad (4.9)$$

$$d_i + d_j \leq CL_k, \forall k \in K$$

$$L_i \leq u_{ik} \leq CL_k \quad \forall i \in C, \forall k \in K \quad (4.10)$$

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

$$y_{ik} \in \{0, 1\} \quad \forall i \in N, \forall k \in K$$

The objective (4.1) of the model is to minimize the total distance travelled by all the vehicles. The constraints of (4.2) to (4.5) are the vehicle capacity constraints. Expression (4.2) for each k forms each vehicle's width constraint. Expression(4.3) is a vehicle's length constraint. Expression(4.4) is the vehicle height constraint. Expression(4.5) is a vehicle's mass constraint. Expression(4.6) and Expression(4.7)

ensure that every customer is assigned to only one truck, with the exception of the depot to be visited by all of the m vehicles. The set of constraints (4.8) imply that a vehicle visits a customer and leaves that customer. The set of constraints (4.9) and (4.10) are the subtour elimination constraints of the travelling Salesman Problem (TSP). That set of constraints imposes both the capacity and the connectivity of CVRP as proposed by Miller, Tucker and Zemlin (1960).

In this model, the capacity constraints are based on four dimensions. These are the length, the height, the width and weight constraints. The length constraint dominates the other capacity constraints, based on the experience in the glass industry the length turns out to be the most important restriction in loading a vehicle. As discussed in Section 3.2, if a plate of glass is longer than the height of the tripod, the workers can increase the height of the tripod by adding some more cushions. Also width of any customer's orders is discussed in the ordering algorithm, the instruction is that if a customer's demands exceeds the truck's demand then a new column is allocated next to previous and the lengths are added together. Therefore, the width and the height constraints turn out to have larger slacks. In addition, the weight constraint has a large slack value, too. The government regulation is that the cargo of a vehicle cannot exceed its hauling capacity of a truck. Past experience of the firm indicates that the total weight of the glass plates cannot exceed the hauling capacity of a truck. Consequently, the height, the width and the weight constraints become redundant. Thus the problem is reduced to one dimensioned (length) capacity constraint instead of four dimensional capacity constraints. The model above now can be restated by

using only one capacity constraint.

$$\min \quad \sum_{i \in N} \sum_{j \in N} d_{ij} \sum_{k \in K} x_{ijk} \quad (4.11)$$

Subject to

$$\sum_{i \in N} L_i y_{ik} \leq CL_k \quad \forall k \in K \quad (4.12)$$

$$\sum_{k \in K} y_{ik} = 1 \quad \forall i \in C \quad (4.13)$$

$$\sum_{k \in K} y_{0k} = m \quad (4.14)$$

$$\sum_{j \in N} x_{ijk} = \sum_{j \in N} x_{jik} = y_{ik} \quad \forall i \in N, \forall k \in K \quad (4.15)$$

$$u_{ik} - u_{jk} + CL_k x_{ijk} \leq CL_k - L_i \quad \forall i, j \in C, i \neq j, \text{ such that} \quad (4.16)$$

$$d_i + d_j \leq CL_k, \forall k \in K$$

$$L_i \leq u_{ik} \leq CL_k \quad \forall i \in C, \forall k \in K \quad (4.17)$$

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

$$y_{ik} \in \{0, 1\} \quad \forall i \in N, \forall k \in K$$

Note that the vehicles are assumed to be identical, then the decision variable for the constructed model above is revised and the model can be reconstructed as in (4.18) to (4.24).

Let

$$x_{ij} = \begin{cases} 1 & \text{if customer } j \text{ is immediately visited after customer } i \\ 0 & \text{otherwise} \end{cases}$$

,

u_i = the load of the vehicle after visiting customer i .

and the mathematical formulation for the revised model is;

$$\min \quad \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij} \quad (4.18)$$

Subject to

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in C \quad (4.19)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in C \quad (4.20)$$

$$\sum_{i \in N} x_{i0} \leq m \quad (4.21)$$

$$\sum_{j \in N} x_{0j} \leq m \quad (4.22)$$

$$u_i - u_j + CLx_{ij} \leq CL - L_i \quad \forall i, j \in C, i \neq j, \text{ such that} \quad (4.23)$$

$$d_i + d_j \leq CL$$

$$L_i \leq u_i \leq CL \quad \forall i \in C \quad (4.24)$$

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

The model above (4.11) to (4.17) is solved by the help of GAMS 20.2 optimizer tool on a computer with an Intel Centrino 1.7 processor. The GAMS model is provided in Appendix B.

CHAPTER 5

Solution Methods

In order to solve the proposed model of Chapter 4 an optimizer tool is needed. GAMS 20.2 is used for an optimizer tool and tried to solve the integer problem. The resource limitations of the GAMS optimization tool and the structure of the loading problem unfortunately cannot allow solving the larger size problems. Therefore alternative methods for solving the proposed model are required for tackling the large sizes problems. Therefore, an exact algorithm (Section 5.1) and a heuristic method (Section 5.2) are proposed.

5.1 An Exact Algorithm for the Problem

Each day the delivery and transportation problem needs to be solved for the transport and delivery of daily customers' orders. An alternative method to solve the problem is the proposed exact algorithm. When the number of customers increase, the constructed model of Chapter 4 cannot be solved. An alternative method is an exact algorithm which is a kind of set-covering-based algorithm for the CVRP suggested by Balinski and Quandt (1964). Also for a detailed review of the set-covering-based algorithms for the CVRP, one can refer to Bramel and Simchi-Levi (2002).

Other set-covering-based methods are summarized as follows. First step is to solve the linear relaxation of the set covering problem by using column generation (CG) procedure without enumerating all possible routes. The solution of the relaxation problem is then used as a lower bound on the optimal integer problem. Finally, in the space of the columns generated an integer solution is obtained by using the branch-and-price algorithm.

Set Covering Problem

Let C = the set of customers, $C = \{1, 2, 3, \dots, n\}$

\mathcal{R} = the set of all feasible routes, $\mathcal{R} = \{1, 2, \dots, R\}$

d_r = the total distance travelled of route r , $r \in \mathcal{R}$

k = number of delivery vehicles, $K = \{1, 2, \dots, m\}$

$$\alpha_{ir} = \begin{cases} 1 & \text{if customer } i \text{ is served in route } r \\ 0 & \text{otherwise} \end{cases}$$

where $i \in C$, $r \in \mathcal{R}$ and the decision variable is;

$$x_r = \begin{cases} 1 & \text{if route } r \text{ is in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

where $r \in \mathcal{R}$. The set covering mathematical formulation is as follows;

$$\min \quad \sum_{r \in \mathcal{R}} d_r x_r \quad (5.1)$$

Subject to

$$\sum_{r \in \mathcal{R}} \alpha_{ir} x_r \geq 1 \quad \forall i \in \mathcal{C} \quad (5.2)$$

$$\sum_{r \in \mathcal{R}} x_r \leq k \quad (5.3)$$

$$x_r \in \{0, 1\} \quad \forall r \in \mathcal{R}$$

The objective function (5.1) of the set covering problem is to select a minimum-distance set of feasible routes. Constraint (5.2) each customer is served in one route and Constraint (5.3) refers at most k number of delivery vehicles can be used. Since our distance matrix d_{ij} satisfies the triangular inequality, each customer is visited exactly ones in the optimal solution whether the Constraint (5.2) is an inequality constraint.

Linear Programming Relaxation of the Set Covering Problem

Without enumerating all the routes, the column generation algorithm is used to solve the LP relaxation of the set covering problem. The solution to this LP relaxation is then used to obtain if there are any routes not included in the formulation. This is the column generation step. To understand if there is a route that should be included in the formulation a simpler optimization problem is solved by using the optimal dual variable values. Then the LP relaxation problem is resolved. This procedure stops when no additional routes are found which can reduce the objective function. After all an optimal solution to the relaxation problem is found.

First a partial set of routes are enumerated where $R' \subseteq \mathcal{R}$. The linear relaxation of

the set covering problem is formulated as;

$$\min \quad \sum_{r \in \mathcal{R}'} d_r x_r \quad (5.4)$$

Subject to

$$\sum_{r \in \mathcal{R}'} \alpha_{ir} x_r \geq 1 \quad \forall i \in C \quad (\Pi_i) \quad (5.5)$$

$$\sum_{r \in \mathcal{R}'} x_r \leq k \quad (\theta) \quad (5.6)$$

$$x_r \geq 0 \quad \forall r \in \mathcal{R}'$$

Let x^* be the optimal solution to the LP relaxation problem of the set covering problem, and Π^* be the corresponding optimal dual variables associated with the Constraint (5.5) as shown in the model where $\Pi^* = \{\Pi_1^*, \Pi_2^*, \dots, \Pi_n^*\}$. Also θ^* be the corresponding optimal dual variable associated with the Constraint 5.6.

The dual of the linear relaxation of the set covering problem is;

$$\min \quad \sum_{i \in C} \Pi_i - K\theta \quad (5.7)$$

Subject to

$$\sum_{i \in C} \alpha_{ir} \Pi_i - \theta \leq d_r \quad \forall r \in R \quad (5.8)$$

$$\Pi_i \geq 0 \quad \forall i \in C$$

$$\theta \geq 0$$

If the dual prices (variables) of the LP relaxation problem satisfies the constraints of the dual of the LP relaxation, then the LP relaxation of the set covering problem is optimal. So we are looking whether optimal (Π^*, θ^*) is dual feasible. As long as the dual feasibility is satisfied, the optimal solution for the LP relaxation of the set covering (primal) problem with set of routes R' is also optimal for the one with the set of routes \mathcal{R} . In order to check the dual feasibility, it is enough to find a column (route)

$r \in R \setminus \mathcal{R}'$. So if we can find a column which violates Constraint 5.7 one should add the corresponding column (or route) to set \mathcal{R}' and should continue the CG procedure. After all the CG algorithm is utilized to identify a feasible route $r \in \mathcal{R}$ that violates 5.7.

Column Generation Algorithm

Lets define d_r^* to be the reduced cost of column r , and

$$d_r^* = d_r + \theta^* - \sum_{i \in S_r} \Pi_i^* \quad (5.9)$$

for each $r \in R$ where S is the set of nodes. Also define $L(S) = \sum_{i \in S} L_i$ for any $S \subseteq C$. Then the column generation problem is;

$$d_{min}^* = \min \{ d_r^* : L(S_r) \leq CL \} \quad (5.10)$$

In order to solve CG, the problem is modelled as a shortest path problem. At each step updated distances (d_{ij}^u) are used for the corresponding shortest path problem rather than using the original ones (d_{ij}). Then the distances are updated by using the following expression;

$$d_{ij}^u = d_{ij} + \Pi_i^*/2 + \Pi_j^*/2 \quad (5.11)$$

So in terms of edge distances d_r can be represented as follows;

$$d_r = \sum_{(i,j) \in r} d_{ij} \quad (5.12)$$

similarly, d_r^*

$$d_r^* = \sum_{(i,j) \in r} d_{ij}^u \quad (5.13)$$

Note that, with this formulation $d_r^* = d_r - \sum_{i \in S_r} \Pi_i^*$. This is a slight modification of the Expression (5.9). Since θ is a fixed cost added to each route while selecting the

minimum distance route (5.10), θ can be eliminated. But when we are looking for the optimality we have to consider it. The elimination of θ allows modeling the CG problem as a shortest path problem (otherwise we should distribute the fixed cost to each edge which is meaningless).

The corresponding shortest path problem may have negative cycles which is known to be \mathcal{NP} -hard. We are going to solve this problem by using a kind of dynamic programming approach (DP). Note that Desrochers et al. (1992) proposed a branch-and-bound algorithm to solve the CG problem. To generate a lower bound for the CG they used dynamic programming. Our dynamic programming approach is different from their algorithm in the sense that we used DP to solve the CG rather than obtaining a lower bound for CG. Another note for the reader is the following; we used branch-and-bound (price) to solve the set covering problem. But, they use branch-and-bound only for to solve CG. The dynamic programming recursion is as follows.

$$f_n(j) = \min_i \{f_{n-1}(i) + d(i, j) : L(n-1, i) + L(j) \leq L\} \quad (5.14)$$

where $f_n(j)$ is the minimum distance of visiting customer j as the n^{th} customer, and $L(n, j)$ is the length of the total load of a truck which visits customer j in the n^{th} order with minimum distance. Also we should keep the information of the predecessor customers that achieves the minimum distance routes. The information about the predecessors are stored by the Expression (5.15).

$$P_n(j) = \arg \min_i \{f_{n-1}(i) + d(i, j)\} \quad (5.15)$$

At each stage the value of $P_n(j)$ is stored as soon as $f_n(j)$ is updated.

Note that, $L(n, j)$ is also updated in each step the following identity.

$$L(n, j) = L(n - 1, P_n(j)) + L_j \quad (5.16)$$

Branch-and-Price Algorithm

We solved the LP relaxation of the set covering problem by using CG algorithm until now. In order to obtain an optimal integer solution, we have to use the branch-and-price algorithm. Solving the LP relaxation of the set covering problem for each subproblem (branch) in a branch-and-bound procedure yields an integer solution for the set covering problem.

A branch-and bound procedure in which additional columns are generated/added at each subproblem of the branch-and-bound tree is proposed which is known as branch-and-price algorithm. As in Desrochers et al. (1992), we branch on the edge variables (e.g. $x_{ij} = 0$ or 1 in the model 4.18 to 4.24 below). When $x_{ij} = 1$, d_{ij} in the dynamic programming algorithm is set to $-\infty$. When $x_{ij} = 0$, d_{ij} in the dynamic programming algorithm is taken as ∞ .

The flow chart of the set-covering-based exact algorithm is shown in Figure 5.1. The algorithm is coded in MATLAB 6.5 and GAMS 20.2 on a computer with an Intel Centrino 1.7 processor. The source code of the exact algorithm is given in Appendix C. Also the GAMS code of the set covering model solved in the exact algorithm is given in Appendix D. The numerical results are presented in Chapter 6.

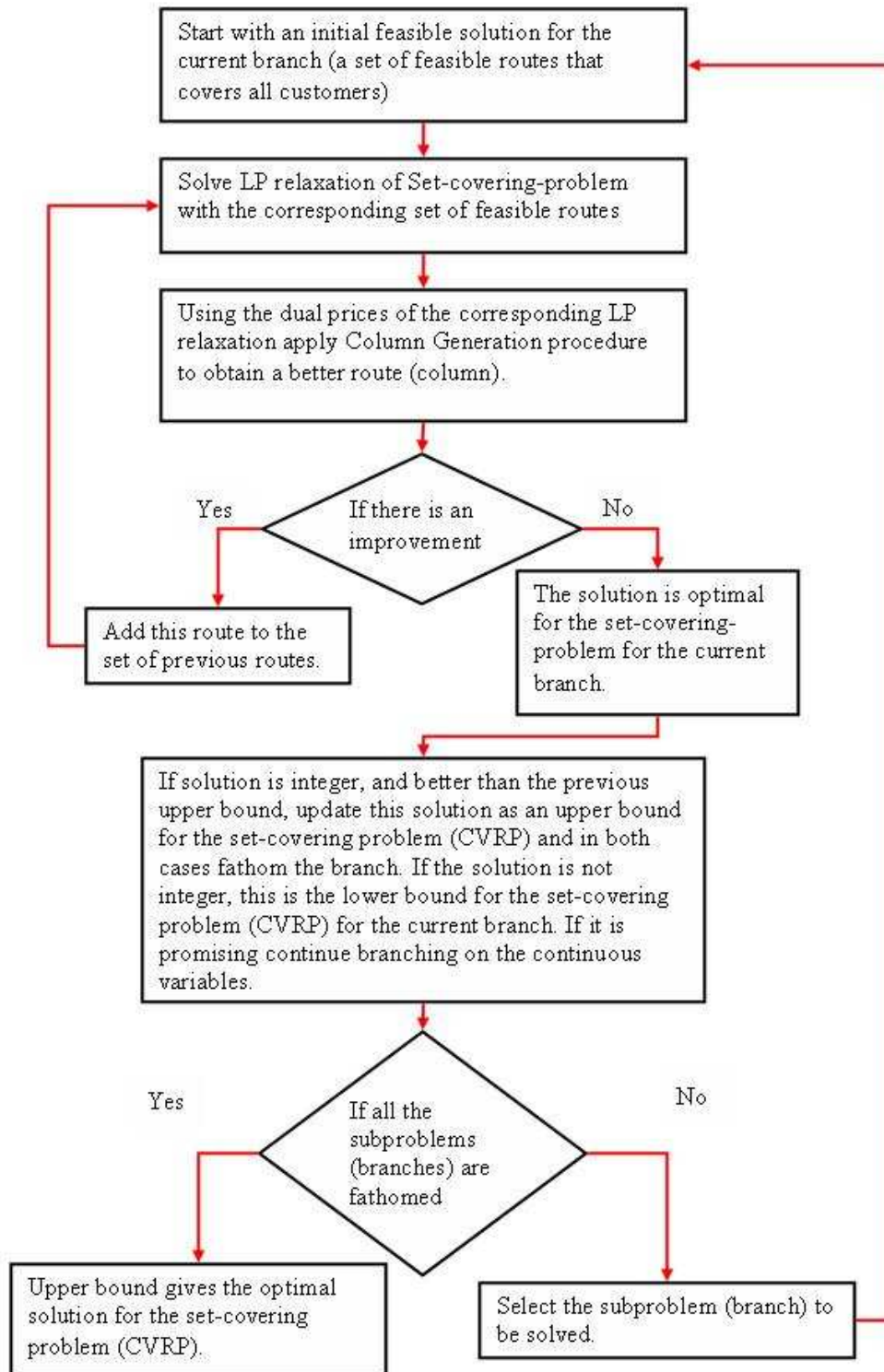


Figure 5.1: The Flow Chart Of The Set-Covering-Based Exact Algorithm

5.2 A Heuristic Approach for the Problem

On further proposed method for solving the model of Chapter 4 is a heuristic, namely the *Generalized Petal Algorithm*. This algorithm is divided into two parts. First part of the Generalized Petal Algorithm consists of The Generalized Petal Heuristics (Section 5.1.1). The second part consists of The Improvement Heuristics (Section 5.1.2).

5.2.1 The Generalized Petal Heuristics

The proposed heuristic generates a set of good vehicle routes by using the generalized petal procedure. Then the optimal route of all these routes is selected by using the set partitioning algorithm.

The petal procedure begins with selecting an initial sequence. The proposed heuristic generates 1-petals using *Traveling Salesman Heuristics* (TSP) which is referred to as generalized petals (Foster and Ryan,1976). TSP heuristics are applied by starting with an arbitrary customer to assign all customers a number from 1 to n representing a sequenced TSP tour. Each feasible subsets taken from this order is a *petal*. A petal is feasible if the length of plates of glasses delivery on the routes does not exceed the length of the vehicle. Such a TSP route for a petal is called a *petal route* (Ryan et all (1993)).

Given a set of petals, an optimal combination can be selected by solving a set partitioning problem. If petals are constructed by following a certain sequence, then

the LP relaxation of the set partitioning problem is totally unimodular. This means the optimal integer solution can be found just solving the LP relaxation of the set partitioning problem (Foster and Ryan (1976) and Ryan et al (1993)).

One of the reason of using TSP heuristics to generate petal routes instead of radial order or sweep procedure is to create good and various sequences. Also in general the cost of a petal route is defined by using the corresponding TSP petal routes if radial order or sweep procedure applied first. But if a TSP procedure is first applied the cost of a petal route is known directly from the petal itself. This is one other reason for applying TSP first.

Four TSP heuristics are used to generate petal routes by combining them together. These are Nearest Neighborhood Heuristic (NNH), Nearest Insertion Heuristic (NIH), Farthest Insertion Heuristic (FIH), Cheapest Insertion Heuristic (CIH). they are all known to be construction heuristics. The construction heuristics stop when a solution is found and never try to improve the solution. A brief explanation of these heuristics are given next.

Nearest Neighborhood Heuristic (NNH)

NNH is perhaps the simplest and most straightforward TSP heuristic. The logic for that algorithm is always to visit the nearest city. This heuristic runs in $O(n^2)$. First a random city is selected to start. Then an unvisited city is found and visited. It stops when there are no unvisited cities remaining.

Nearest Insertion Heuristic

In this heuristic, a node with the shortest distance to a tour node is inserted.

Farthest Insertion Heuristic

The node is inserted to a tour node whose minimal distance is maximum.

Cheapest Insertion Heuristic

In the CIH, any city is selected first to start and its closest neighbor is found. Then a subtour is created to join those two cities. Next an arc is replaced in the subtour by combining those two arcs. This procedure is repeated until a tour is obtained.

2-Opt Heuristic

We used 2-Opt Heuristics method is implemented immediately after applying the four construction TSP heuristics to improve the sequence obtained. 2-Opt Heuristic is an improvement heuristic. The improvement heuristics attempt improving a solution which was generated by some construction heuristic.

Given an initial tour, 2-Opt heuristic eliminates two edges and reconnects the two resulting subpaths in some other way. The 2-Opt heuristic runs in $O(n^2)$. A scheme for 2-Opt heuristic is shown in Figure 5.2.

Figure 5.2 shows how easily the savings for the distance of nodes (i, k) can easily be calculated as;

$$\text{Payoff } (i, k) = d_{ij} + d_{lk} - d_{ik} - d_{jl} \quad (5.17)$$

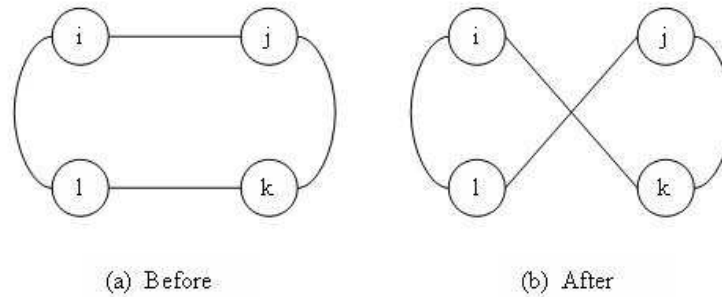


Figure 5.2: 2-Opt

The logic is to swap the corresponding edges and to calculate all of the swapped edges' payoffs by using the Equation (5.17). The swaps with the maximum gains are exchanged.

Set Partitioning Algorithm

After finding and improving the TSP route, all the feasible petals are found. Then all feasible petal routes are enumerated. A feasible petal is a feasible route starts and ends at the depot. Since TSP is solved first, subsequently the distances of the petals are calculated. In order to select an optimal tour, the problem is reformulated as a set partitioning problem. The parameters of the set partitioning problem are as follows.

Let

C = the set of customers, $C = \{1, 2, 3, \dots, n\}$

\mathcal{P} = the set of all feasible petals, $\mathcal{P} = \{1, 2, \dots, P\}$

d_p = the distance travelled of petal p

k = number of delivery vehicles, $K = \{1, 2, \dots, m\}$

$$\alpha_{ip} = \begin{cases} 1 & \text{if customer } i \text{ is served in petal } p \\ 0 & \text{otherwise} \end{cases}$$

where $i \in C$, $p \in \mathcal{P}$, and the decision variable is;

$$x_p = \begin{cases} 1 & \text{if petal } p \text{ is in the optimal solution} \\ 0 & \text{otherwise} \end{cases}$$

where $p \in \mathcal{P}$. The set partitioning mathematical formulation is as follows;

$$\min \quad \sum_{p \in \mathcal{P}} d_p x_p \quad (5.18)$$

Subject to

$$\sum_{p \in \mathcal{P}} \alpha_{ip} x_p = 1 \quad \forall i \in C \quad (5.19)$$

$$\sum_{p \in \mathcal{P}} x_p \leq k \quad (5.20)$$

$$x_p \in \{0, 1\} \quad \forall p \in \mathcal{P}$$

The objective function (5.18) of the set partitioning problem selects a minimum-distance set of feasible petals. Constraint (5.19) implies each customer is served in one route and Constraint (5.20) requires that at most k number of delivery vehicles can be used. The solution of the set partitioning problem gives the optimal petals, in other words optimal vehicle routes and their distances.

To solve the set partitioning problem MATGAMS is utilized to obtain the VRP solutions. For each TSP sequences, the proposed heuristic finds one VRP solution. There is one TSP sequence immediately after applying the improvement heuristic

(2-Opt) to, for instance, NNH. Then petal algorithm is applied to the improved TSP sequence which comes from NNH to find all feasible petal routes. Then the set partitioning problem is run to find the VRP solution of the found petal routes. This is done for all of the four construction heuristics. Finally, the best VRP solution of all is selected.

5.2.2 The Improvement Heuristic for VRP

In Section 5.2.1 the generalized petal heuristic approach is explained. In this Section an improvement VRP heuristic is applied to the best found optimal VRP obtained in Section 5.2.1.

A variation of Multi-route improvement heuristic is used to improve the preceding vehicle routes instead of Single-route improvements in the previous subsection. Multi-route improvement heuristics provide descriptions of multi-route edge exchanges for the VRP. String Exchange (SE) performs better than the other multi-route improvement heuristics in the literature (Toth and Vigo, (2002)). Thus, the selected String Exchange method is preferred for our proposed heuristic to improve the VRP.

String Exchange Heuristic

Figure 5.3 shows how String Exchange is applied to a VRP. Two strings (or chains) of at most k vertices are exchanged between two routes where k represents the string length, $k = 1$ for our problem.

Let's exchange i from one route and j from the other route in Figure 5.3. Then the

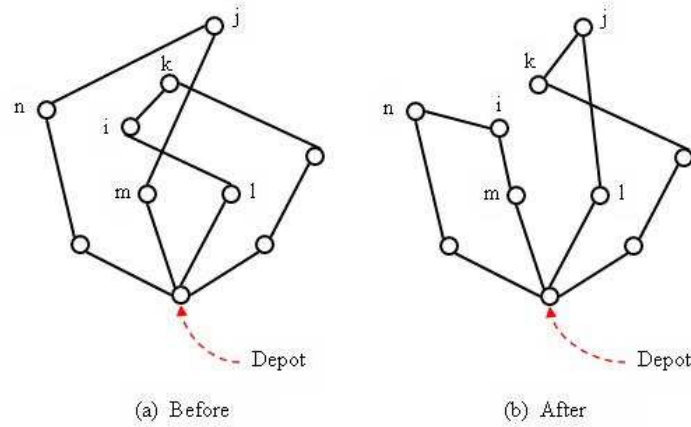


Figure 5.3: String Exchange

payoff matrix can be calculated as;

$$\text{Payoff } (i, j) = d_{ik} + d_{li} + d_{jm} + d_{nj} - d_{im} - d_{in} - d_{jk} - d_{lj} \quad (5.21)$$

The logic for the string exchange is to first evaluate all possible swapped nodes' payoffs between a pair of routes by using the Expression (5.21), and then select the maximum payoff for the exchange procedure.

The steps of generalized petal heuristic algorithm is shown below;

Step 1 : Initialization of system parameters

Demand and coordinates of the customers, capacity of the trucks, maximum number of trucks.

The distance matrix is calculated using the coordinates of the customers. (All the distances are Euclidean distances)

Step 2 : For each TSP construction heuristic (NNH, NIH, FIH, CIH) do the following (outer loop)

Step 3 : For each $i \in C$ do the following: (inner loop)

Step 3.1 : Apply NNH starting from $i \in C$ to find a TSP solution.

Step 3.2 : Apply 2-Opt procedure to improve the TSP tour of Step 2.1.

Step 3.3 : Apply the generalized petal heuristic. Note that the TSP solution of Step 3.2 is used as the sequence to be followed during the generalized petal procedure.

Step 3.4 : Solve a set partitioning problem to select the optimal petal routes using the solution of Step 3.3. Note that this is the first time in the algorithm that a (feasible, probably a good) VRP solution is obtained.

Step 3.5 : Apply string exchange procedure to improve the VRP solution of Step 3.4.

Step 4 : Stop when the inner and outer loops are completed.

The flow chart of the generalized petal algorithm is shown in Figure 5.4. The source code of the generalized petal heuristic can be seen in Appendix E. Also the GAMS code of set partitioning model solved in the generalized petal heuristic is given in Appendix F.

The results of numerical experimentations and the comparisons of the proposed heuristic to others are presented in Chapter 6.

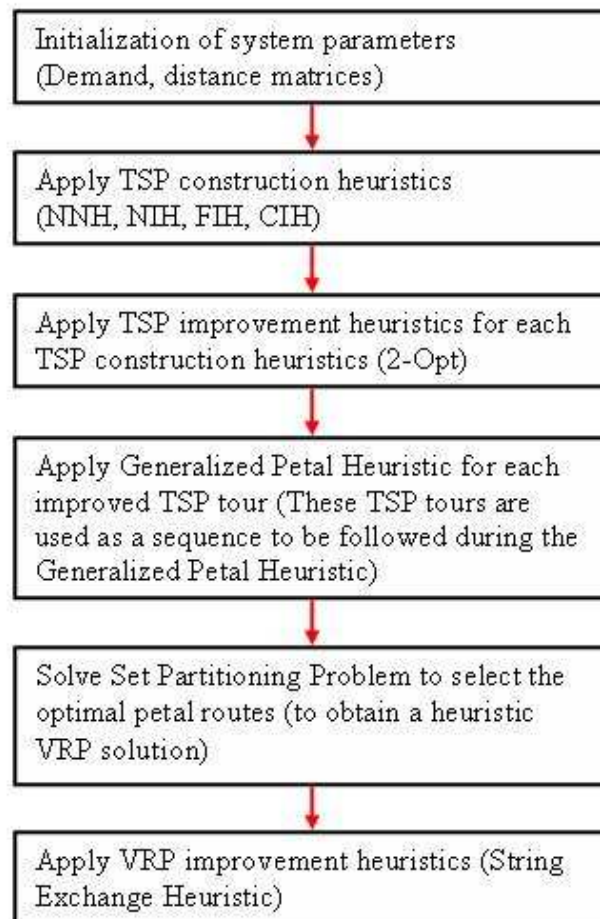


Figure 5.4: The Flow Chart Of The Generalized Petal Heuristic

CHAPTER 6

Numerical Experimentation

This Chapter presents the numerical results of the solutions of these following three methods; The IP model of Chapter 4, the exact solution method (Section 5.1) and the proposed heuristic (Section 5.2). The benchmark problems described by Christofides et al. are used to test the performance of the exact algorithm and the proposed heuristic. After that the algorithms are used to find the solution of the problem.

Intel Centrino 1.7 processor computer is used to find the solutions of all of the proposed methods. For the optimization, parts of the GAMS 20.2 is used as an optimization tool. MATLAB 6.5 is used for the coding part of all of the methods described. The disadvantage of using the MATLAB and MATGAMS is they use CPU so much, thus the CPU times for the methods are increasing, and this directly affects the performance of the solution methods.

Even for the proposed IP models, Equations (4.1) to (4.10) and the other model Equations (4.11) to (4.17), upto ten (10) customers can be solved by using the GAMS model in Appendix B because of the structure of the CVRP. Thus it is clear that CVRP models and its solution methods cannot satisfy the requirements for the firm's problem. For this reason we need alternative solution methods to solve the problem.

An alternative solution method described in Section 5.1 is an exact solution method. This exact method is a variant of set-covering-based algorithm with branch-and-price algorithm to solve the proposed model. By using the benchmark problem it seen that exact algorithm can solve the problems upto twenty one (21) customers. It is observed that the exact algorithm cannot solve more than 21 customers and when the CPU times are based on it is seen that this algorithm performs very slow. The computational comparison of the benchmark problems solutions and the exact algorithm solutions and the CPU times in seconds are summarized in Table 6.1.

Table 6.1: Comparison Of The Exact Algorithm With The Optimum

		Christofides et al.		Exact Algorithm		
Prob. Set	Customer	Optimal	Routes	Optimal	Routes	CPU Time
E-n13-k4	12	247	4	247	4	104.03
E-n22-k4	21	375	4	375	4	310.2

Consequently, another alternative method is proposed to solve the problem, hence a heuristic method is tested. In Section 5.2, the generalized petal algorithm is mentioned which solves TSP first and after finds all of the feasible routes, finally improves the VRP solution obtained by applying set partitioning model to the petal routes. The computational comparison of the generalized petal algorithm to the optimal twelve benchmark problems are given in Table 6.2.

As seen from the Table 6.2 the generalized petal algorithm solves the problem with 100 customers and more. Also in this table the CPU times in seconds of the proposed heuristic is provided. In the TSP heuristic column of the table the TSP construction

Table 6.2: Comparison Of The Gen.Petal Algorithm With The Optimum Solution

Prob. Set	Christofides et al.		Generalized Petal Algorithm			
	Optimal	Routes	Solution	Routes	CPU Time	TSP Heu.
E-n13-k4	247	4	249	4	27.90	NNH
E-n22-k4	375	4	377	4	39.85	NNH
E-n23-k3	569	3	576	3	48	NNH, FIH
E-n30-k3	534	3	571	3	67.71	FIH
E-n33-k4	835	4	856	4	76.09	NNH
E-n51-k5	521	5	557	6	153.06	NNH
E-n76-k7	682	7	735	7	479.31	NNH,FIH
E-n76-k8	735	8	805	10	371.16	FIH
E-n76-k10	830	10	901	11	278.22	CIH
E-n76-k14	1021	14	1101	16	215.31	FIH
E-n101-k8	817	8	898	8	1473.7	NNH
E-n101-k14	1071	14	1171	14	567.42	NNH

heuristics are shown with the best VRP solution.

The solutions of the proposed heuristic are far from the optimal solutions even from the other heuristic solutions. The reason for that is to use the MATLAB for coding. Also the Sweep method (Gillet and Miller (1974)) uses 3-opt, 1-petal (Foster and Ryan (1976)) and 2-petal (Renaud et all (1996)) method uses 4-opt improvement heuristics. Note that in our heuristic 2-opt improvement heuristic is used instead of 3-opt or 4-opt. In Table 6.3 some of the heuristic solutions are given in km to compare the solutions with generalized petal heuristic in kilometers.

Table 6.3: Best Solution Comparison Of The Gen.Petal With Other Heuristics

Prob. Set	Clark&Wright	Sweep	1-Petal	2-Petal	Gen.Petal	Optimum
E-n51-k5	578.64	532	531.90	524	557	524
E-n76-k10	900.26	884	885.02	854.09	901	830
E-n101-k8	886.83	851	836.34	830.40	898	817

In Table 6.4 the comparison of the CPU times in seconds for the heuristics are shown.

Table 6.4: CPU Time Comparisons Of The Gen.Petal With The Other Heuristics

Prob. Set	Sweep	1-Petal	2-Petal	Gen.Petal
E-n51-k5	0.12	0.1	0.76	153.06
E-n76-k10	0.17	0.07	0.52	278.22
E-n101-k8	1.18	0.32	3.84	1473.7

After the comparisons of the exact algorithm and the proposed heuristics to the benchmark problems in the literature, the daily delivery and transportation problem of the firm is solved both using the methods described in Chapter 4 and 5.

For a given daily data, the solutions are tested by comparing our solutions with the solution given from the firm. It is known from the given data that there are 21 customers for delivery. The height, the length and the width of each glass is known for any customer. Before loading the vehicles the sequencing the customers' orders algorithm is applied to all of the customers and obtained the length, the height, the width and the weight of the customers. The solutions of all customers can be seen in Appendix G. This is the data (Section 3.2) input for the model to be solved. The

distance matrix in km is calculated by the help of a map, and this matrix satisfies the triangular inequality and is given in Appendix H.

The CVRP model described in Chapter 4 is used, but it cannot be solved for the larger customer numbers. Unfortunately the exact algorithm cannot solve larger size problems. The proposed heuristic finds a solution in 45.297 seconds (CPU time). The total distance travelled is 233.6 km and only 2 vehicles are needed to load the customer orders. There are many alternative solutions. Two of the alternative routes are given below.

Vehicle 1 : 0-3-2-12-10-11-16-6-5-9-7-4-0

Vehicle 2 : 0-20-19-18-8-1-14-13-17-21-15-0

or

Vehicle 1 : 0-9-7-4-16-6-5-2-11-12-10-3-0

Vehicle 2 : 0-15-21-17-13-14-20-19-18-8-1-0

They enumerate the customers first and then load to the vehicles from the first customer on the list to the last. Any clustering or routing operation is not applied for the delivery and transportation problem. The solution of the firm is to load the customer orders into two vehicles. The vehicles travel a total distance of 331.9 km. The routes for the firm are given below;

Vehicle 1 : 0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-0

Vehicle 2 : 0-17-18-19-20-21-0

The difference between the solution of the firm and the proposed heuristic is 98.3 km. This means there is a savings of 98.3 km by the generalized petal algorithm in comparison to the firm's current strategy.

CHAPTER 7

Conclusion

This thesis attempts solving a delivery and a transportation problem of a glass manufacturing firm. The problem of the firm is to deliver the customer orders to the customers on time. The customers orders are various sizes of plates of treated glass. They first need to load the glass plates into the delivery vehicles. The glass plates must be loaded carefully because of the brittleness structure of the glass. An important aspect for loading is the capacity of a vehicle. The customer orders should not exceed the capacity of a vehicle. In this problem, the capacity of a vehicle's expressed in four dimension. These are the length, the width and the height of a vehicle. Also the weight capacity of the vehicle is the fourth dimensions. The total load of a vehicle should satisfy all of these restrictions.

This problem is modelled as a CVRP. The length, the width, the height and the weight of each customer should be known for checking the capacity constraints. Each customer's order is arranged in some fashion by a sequencing algorithm for loading. This heuristic algorithm is developed using the workers' experiences. The importance of this algorithm is the output of this algorithm for each customer is used as an input of CVRP and the other solution methods. The output of the glass plates arranging algorithm gives the length, the width, the height and the weight of each customer.

And this data are used for the demand data of the customers or for the capacity constraints of the vehicles.

After finding the capacities of each customers the CVRP model is constructed and attempted solving by using an optimizer tool. This optimizer tool is selected as GAMS 20.2, and used for to solve this problem. Because of the structure of the problem and the resource limits of the GAMS, the model cannot be solved. Unfortunately upto ten customers can be solved. For this reason other methods are needed to solve the problem.

An alternative solution method used to solve the delivery and the transportation problem is an exact algorithm which is a set-covering-based algorithm. Briefly in this algorithm the CVRP is modelled as a set-covering problem. By using the branch-and-price algorithm the set-covering problem is used to solve. The relaxation of the set covering problem is used and tried to solve by using the column generation algorithm to find a lower bound on the set-covering problem. The branch-and-price algorithm is used to find an integer solution to the found the solution of the set-covering relaxation problem. This algorithm is coded by using MATLAB 6.5 and by using MATGAMS. Because of using MATLAB the algorithm performs worse. Upto twenty one customers can be solved in a long time period. Thus an other alternative method is proposed.

The other alternative method is an heuristic approach. This heuristic approach is a generalized petal algorithm. Four TSP construction heuristics are used at the same time to find a TSP tour to start the generalized petal algorithm. These heuristics are

Nearest Neighborhood, Nearest Insertion, Farthest Insertion, and Cheapest Insertion. They are used to find the starting order to the petal algorithm. After finding the TSP tour then a TSP improvement heuristic (2-Opt) is applied to improve the TSP tour after the improvement the petal algorithm is applied to find all feasible petals. The set-partitioning algorithm finds the VRP routes among them the best is chosen. Then an VRP improvement heuristic (String Exchange) is applied to improve the VRP route. Four heuristics are utilized simultaneously. Subsequently, the algorithm selects the best VRP. The best VRP can come from either NNH or FIH or any of them. This heuristic is also coded by using MATLAB 6.5 and MATGAMS. Unfortunately, using MATLAB causes the heuristic performing worse than the proposed other heuristics in the literature. A number of hundred customers can be solved by using the proposed heuristic in a longer period of time than the other heuristics in the literature.

The exact algorithm and the proposed heuristic is tested by using a benchmark problems described by Christofides et al. The performance of the exact algorithm and the proposed heuristic are worse than the exact and heuristic methods in the literature. One of the reasons of this is to use MATGAMS for coding. The other reason is that the heuristics in the literature used 3-opt or 4-opt improvement heuristics, but we used only 2-opt improvement heuristic in this thesis.

Then the proposed model of delivery and the transportation problem is solved by using, the exact algorithm and the proposed heuristic. A daily given data is used which they have to deliver twenty one customers in that day. First of all the length, the width, the height and the weight of each customer is determined by using the glass plate arranging algorithm. Then the constructed CVRP model is used but cannot

solve the problem. After the exact method is applied and it cannot solve the problem. Then the proposed heuristic is used and finds a solution with two vehicles at a total traveling distance of 233.6 km. The firm uses currently also two vehicles at a total distance of 331.9 km to transport and distribute the customer orders. Therefore, the proposed model to the problem reduces the total traveling distance by 98.3 km by using the generalized petal heuristics. This is a considerable daily savings for the firm in terms of traveling distances and consequently, the fuel expenses.

REFERENCES

- [1] **Achuthan, N.R., Caccetta, L. and Hill, S.P.**, (2003), 'An Improvement Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem', *Transportation Science*, Vol 2(37), pg. 153-169.
- [2] **Agarwal, Y., Mathur, K. and Salkin, H.M.**, (1989), 'A Set-Partitioning-Based Exact Algorithm for the Vehicle Routing Problem', *Networks*, Vol(19), pg. 731-749.
- [3] **Altinkemer, K. and Gavish, B.**, (1991), 'Parallel Savings Based Heuristic for the Delivery Problem', *Operations Research*, Vol(12), pg. 456-469.
- [4] **Balinski, M. and Quandt, R.**, (1964), 'On An Integer Program for A Delivery Problem', *Operations Research*, Vol(12), pg. 300-304.
- [5] **Beasley, J.E.**, (1983), 'Route First Cluster-Second Methods for Vehicle Routing', *Omega*, Vol(11), pg. 403-408.
- [6] **Bixby, A., Coullard, C. and Simchi-Levi, D.**, (1997), 'The Capacitated Prize-Collection Travelling Salesman Problem', Working paper, Department of Industrial Engineering and Engineering Management, Northwestern University, Evanston, IL Vol(11).
- [7] **Blasum, U. and Hochstattler, W.**, (2002), 'Application of the Branch and Cut Method to the Vehicle Routing Problem', *Operations Research*, Vol(43), pg. 649-660.
- [8] **Bramel, J. and Simchi-Levi, D.**, (1995), 'A Location Based Heuristic for General Routing Problems', *Operations Research*, Vol(43), pg. 649-660.
- [9] **Christofides, N., Mingozi, A. and Toth, P.**, (1979), 'The Vehicle Routing Problem', In Christofides, N., Mingozi, A., Toth, P., and Sandi, C. editors, *Combinatorial Optimization*, Wiley, Chichester, pg. 431-448.

- [10] **Christofides, N., Mingozi, A. and Toth, P.**, (1981), 'Exact Algorithms for the Vehicle Routing Problems, Based On Spanning Tree and Shortest Path Relaxations', *Mathematical Programming*, Vol(20), pg. 255-282.
- [11] **Clarke, G. and Wright, J.V.**, (1964), 'Scheduling of Vehicles From A Central Depot to A Number of Delivery Points', *Operations Research*, Vol(12), pg. 568-581.
- [12] **Desrochers, M. and Vergog, T.W.**, (1989), 'A Matching Based Savings Algorithm for the Vehicle Routing Problem', *Technical Report Cahiers du GERAD G-89-04*, Ecole des Hautes Etudes Commerciales de Montreal, Canada.
- [13] **Desrochers, M., Desrosires, J. and Solomon, M.M.**, (1992), 'A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows', *Operations Research*, Vol(40), pg. 342-354.
- [14] **Gillet, B.E. and Miller, L.R.**, (1974), 'A Heuristic Algorithm for the Vehicle Dispatch Problem', *Operations Research*, Vol(22), pg. 340-349.
- [15] **Fischetti, M., Toth, P. and Vigo, D.**, (1994), 'A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem On Directed Graphs', *Operations Research*, Vol(42), pg. 846-859.
- [16] **Fisher, M.L. and Jaikumar, R.**, (1981), 'A Generalized Assignment Heuristic for the Vehicle Routing Problem', *Networks*, Vol(11), pg. 109-124.
- [17] **Fisher, M.L.**, (1994), 'Optimal Solution of Vehicle Routing Problems Routing Problem Using Minimum k -Trees', *Operations Research*, Vol(42), pg. 626-642.
- [18] **Foster, B.A. and Ryan, D.M.**, (1976), 'An Integer Approach to the Vehicle Scheduling Problem', *Operations Research*, Vol(27), pg. 367-384.
- [19] **Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragao, M., Reis, M., Uchoa, E. and Werneck, R.F.**, (2006), 'Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem', *Mathematical Programming*, Vol(106), pg. 491-511.
- [20] **Hadjiconstantinou, E., Christofides, N., and Mingozi, A.**, (1995), 'A New Exact Algorithm for the Vehicle Routing Problem Based On q -Paths and k -Shortest Paths Relaxations', *Annals of Operations Research*, Vol(61), pg. 21-43.

- [21] **Laporte, G., Mercure, H., and Nobert, Y.**, (1986), 'An Exact Algorithm for the Asymmetric Capacitated Vehicle Routing Problem', *Networks*, Vol(16), pg. 33-46.
- [22] **Laporte, G.**, (1992), 'The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms ', *European Journal of Operational Research*, Vol(59), pg. 345-358.
- [23] **Lin, S. and Kernighan, B.W.**, (1973), 'An Effective Heuristic Algorithm for the Travelling Salesman Problem ', *Operations Research*, Vol(21), pg. 498-516.
- [24] **Lysgaard, J., Letchford, A.N. and Eglese, R.W.**, (2004), 'A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem', *Mathematical Programming*, Vol(100), pg. 423-445.
- [25] **Miller, D.L., Tucker, A.W., and Zemlin, R.A.**, (1960), 'Integer Programming Formulations and Travelling Salesman Problems', *Journal of the ACM*, Vol 7(1), pg. 326-329.
- [26] **Mole, R.H. and Jameson, S.R.**, (1976), 'A Sequential Route Building Algorithm Employing A Generalized Savings Criterion', *Operational Research Quarterly*, Vol(27), pg. 503-511.
- [27] **Ralphs, T.K., Kopman, L., Pulleyblank, W.R and Trotter, L.E.**, (2003), 'On the Capacitated Vehicle Routing Problem', *Mathematical Programming*, Vol(94), pg. 343-359.
- [28] **Renaud, J., Boctor F.F., and Vigo, D.**, (1996), 'An Improvement Petal Heuristic for the Vehicle Routing Problem', *Journal of Operations Research Society*, Vol(47), pg. 329-336.
- [29] **Ryan, D.M., Hjorring, C., and Glover, F.**, (1993), 'Extensions of the Petal Method for Vehicle Routing', *Journal of Operations Research Society*, Vol(44), pg. 289-296.
- [30] **Toth, P. and Vigo, D.**, (2002), 'Integer Programming Formulations and Travelling Salesman Problems', *Journal of the ACM*, Vol 7(1), pg. 326-329.
- [31] **Van Breedam, A.** (1994), 'An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for A Selection of Problems With Vehicle-Related, Customer-Related, and Time-Related Constraints', *Ph.D. dissertation*, University of Antwerp.

- [32] **Wark, P. and Holt, J.**, (1994), 'A Repeated Matching Heuristic for the Vehicle Routing Problem', *Journal of Operational Research Society*, Vol(45), pg. 1156-1167.

APPENDIX A

The Source Code Of Arranging The Glass Plates Algorithm

```
clear;

parameteritem=[194 1241 18 471 1111 18 601 1091 18 1074 1194 36 194 609
18 574 609 18 704 609 18 1074 609 36 ];

widthcapacity=700;

item=parameteritem(:,1:2);

itemsize=size(item);

sequence=[1];

sequencesize=size(sequence);

width=parameteritem(1,3);

for i=2:itemsize(1,1);

j=1;

true=1;
```

```

while (j<=sequencesize(1,1))*true;

item1=i;

item2=sequence(j,1);

if max(item(item1,:))>max(item(item2,:));

if min(item(item1,:))<min(item(item2,:));

sequence=[sequence(1:j-1,:)

item1

sequence(j:sequencesize(1,1),:)];

true=0;

end

end

if max(item(item1,:))<max(item(item2,:));

if min(item(item1,:))<min(item(item2,:));

if (max(item(item1,:))== max(item(item2,:)))+ (min(item(item1,:))<=
min(item(item2,:)))<2;

sequence=[sequence(1:j-1,:)

item1

sequence(j:sequencesize(1,1),:)];

true=0;

end

end

end

if max(item(item1,:))==max(item(item2,:));

if min(item(item1,:))<min(item(item2,:));

```

```

    if (max(item(item1,:)) == max(item(item2,:))) + (min(item(item1,:)) <=
min(item(item2,:))) < 2;

    sequence=[sequence(1:j-1,:);

    item1

    sequence(j:sequencsize(1,1,:))];

    true=0;

    end

    end

    end

    if min(item(item1,:)) == min(item(item2,:));

    if max(item(item1,:)) < max(item(item2,:));

    if (max(item(item1,:)) == max(item(item2,:))) + (min(item(item1,:)) <=
min(item(item2,:))) < 2;

    sequence=[sequence(1:j-1,:);

    item1

    sequence(j:sequencsize(1,1,:))];

    true=0;

    end

    end

    end

    j=j+1;

    end

    if true == 1;

    sequence=[sequence

```

```

item1];

end

sequencesize=size(sequence);

width=width+parameteritem(item1,3);

end

load=0;

sequenceditem=item(sequence(1,1),:);

r=1;

for i=2:sequencesize(1,1);

if load<=widthcapacity

Part(r).sequenceditem=sequenceditem;

Part(r).width=load;

else

r=r+1;

sequenceditem=item(sequence(i-1,1),:);

Part(r).sequenceditem=sequenceditem;

load=parameteritem(sequence(i-1,1),3);

Part(r).width=load;

end

sequenceditem=[sequenceditem

item(sequence(i,1),:)];

load=load+parameteritem(sequence(i,1),3);

end if load<=widthcapacity

Part(r).sequenceditem=sequenceditem;

```

```

Part(r).width=load;

else

r=r+1;

sequenceditem=item(sequence(i,1),:);

Part(r).sequenceditem=sequenceditem;

load=parameteritem(sequence(i,1),3);

Part(r).width=load;

end

overalllength=0;

for i=1:r

sequenceditem=Part(i).sequenceditem;

height(i)=max(max(sequenceditem'));

length(i)=max(min(sequenceditem'));

overalllength=length(i)+overalllength;

end

overallheight=max(height);

overall=[overalllength overallheight];

```

APPENDIX B

The Gams Model Of The Capacitated Vehicle Routing Problem (CVRP)

SETS

i customers /0*21/

k vehicle /1*6/

alias(i,j) ;

SCALAR

m /6/

CWk /1500/

CLk /12000/

CMk /8/

CHk /3000/ ;

PARAMETERS

WIDTH(i) / 1 180, 2 1156, 3 912, 4 130, 5 20, 6 96, 7 80, 8 120, 9 24, 10 48,

11 22, 12 110, 13 8, 14 32, 15 12, 16 648, 17 620, 18 456, 19 180, 20 324, 21 1416/

LENGHT(i) / 1 1074, 2 1100, 3 1194, 4 1830, 5 498, 6 508, 7 459, 8 535, 9
775, 10 377, 11 683, 12 994, 13 445, 14 390, 15 970, 16 654, 17 1127, 18 1180, 19
679, 20 799, 21 2159/

MASS(i) / 1 0.12179, 2 0.61248, 3 0.36913, 4 0.44837, 5 0.00917, 6 0.02808, 7
0.01847, 8 0.02443, 9 0.01853, 10 0.010947, 11 0.027286, 12 0.141719, 13 0.00426,
14 0.02047, 15 0.05134, 16 0.13715, 17 0.33348, 18 0.33807, 19 0.07667, 20 0.12049,
21 1.39296/

HEIGHT(i) / 1 1214, 2 1825, 3 912, 4 2770, 5 936, 6 1051, 7 804, 8 925, 9
1215, 10 1107, 11 2030, 12 1440, 13 490, 14 860, 15 2990, 16 1136, 17 1997, 18
1480, 19 954, 20 1034, 21 2609/ ;

TABLE

DISTANCE(i,j)

Use the distance matrix in Appendix H.

VARIABLES

Z

u(i,k)

u(j,k);

BINARY VARIABLES

x(i,j,k)

y(i,k) ;

EQUATIONS

obj

wid(i,k)

heig(i,k)

leng(k)

mas(k)

customer(i)

totalvehicle(i)

c7(j,k)

c8(i,k)

c11(i,j,k)

c12(i,k)

c13(i,k) ;

obj..Z=e=sum((i,j,k),DISTANCE(i,j)*x(i,j,k));

wid(i,k).. WIDTH(i)*y(i,k)=l=CWk;

heig(i,k)..HEIGHT(i)*y(i,k)=l=CHk;

leng(k)..sum((i),LENGHT(i)*y(i,k))=l=CLk;

mas(k).. sum((i),MASS(i)*y(i,k))=l=CMk;

customer(i)\$ (ord(i) > 1)..sum(k,y(i,k))=e=1;

totalvehicle(i)\$ (ord(i) eq 1)..sum(k,y(i,k))=l=m;

c7(i,k)..sum((j),x(i,j,k))-y(i,k)=e=0;

c8(i,k)..sum((j),x(j,i,k))-y(i,k)=e=0;

c9(i,j,k)\$ (ord(i) > 1 and ord(j) > 1 and ord(i) ne ord(j)

(LENGHT(i) + LENGHT(j);CLk)..u(i,k) - u(j,k) + CLk*x(i,j,k) =l=

CLk - LENGHT(j) ;

c10(i,k)\$ (ord(i) > 1)..LENGHT(i)- u(i,k)=l=0;

c11(i,k)\$ (ord(i) > 1)..u(i,k)- CLk =l=0;


```
MODEL tez /ALL/;
```

```
SOLVE tez MINIMIZING z USING MIP;
```

```
display x.l;
```

```
display y.l;
```

APPENDIX C

The Source Code Of Exact Algorithm

```
clear all;

format long;

t=cputime;

X=[266 235 295 272 301 258 309 260 217 274 218 278 282 267 242 249 230
262 249 268 256 267 265 257 267 242 259 265 315 233 329 252 318 252 329 224
267 213 275 192 303 201 208 217 326 181];

[x1 x2]=size(X);

for i=1:x1

for j=1:x1

if i==j

Distance(i,j)=1000000;

else

Distance(i,j)=round(sqrt((X(i,1)-X(j,1))^2+(X(i,2)-X(j,2))^2));

end

end
```

```

end

end

t=cputime;

Demand=[1074 1630 1728 1830 498 508 459 535 775 377 683 994 445 390 970
654 1127 1180 679 799 4253];

Demand0=[0 Demand];

capacity=12000;

nt=2; [vertexsize,v1] = size(Distance);

vertexsize=vertexsize-1;

branch=1;

solve=1;

fathom=0;

UB=1000000;

count=1;

Demandforsweeper=Demand0;

Demandforsweeper(1,1)=1000000;

i=1;

totaldemand=0;

mindemand=0;

while count<vertexsize+1

j=1;

Pathset(i,j)=1;

while (totaldemand <=capacity)*(mindemand < 1000000)

j=j+1;

```

```

[mindemand mindex]=min(Demandforsweeper);
totaldemand=totaldemand+Demand0(1,mindex);
if totaldemand <=capacity
Pathset(i,j)=mindex;
Demandforsweeper(1,mindex)=1000000;
count=count+1;
PathsetLoad(i,1)=totaldemand;
end
end
Pathset(i,j)=1;
totaldemand=0;
i=i+1;
end
SP(1).zeroelements=[ ];
SP(1).oneelements=[ ];
SP(1).zeroidentifier=zeros(vertexsize+1,1);
SP(1).oneidentifier=zeros(vertexsize+1,1);
SP(1).branchingidentifier=zeros(vertexsize+1);
while solve <=branch
zeroelements=SP(solve).zeroelements;
oneelements=SP(solve).oneelements;
minpath=-1000000;
[s11 s12]=size(oneelements);
FakeDistance=Distance;

```

```

for i=1:s11
    FakeDistance(oneelements(i,1),oneelements(i,2))=0;
    FakeDistance(oneelements(i,2),oneelements(i,1))=0;
end

[s01 s02]=size(zeroelements);

for i=1:s01
    FakeDistance(zeroelements(i,1),zeroelements(i,2))=1000000000;
    FakeDistance(zeroelements(i,2),zeroelements(i,1))=1000000000;
end

[setsize, setsize1] = size(Pathset);

Cost=zeros(setsize,1);

for i=1:setsize
    for j=1:setsize1-1
        if Pathset(i,j+1)>0
            Cost(i)=Cost(i)+FakeDistance(Pathset(i,j),Pathset(i,j+1));
        end
    end
end

for i=1:setsize
    j=2;
    while 1-(Pathset(i,j)==1)
        TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;
        j=j+1;
    end
end

```

```

end

iteration=0;

loop=0;

u2=0;

while (1-loop)*(minpath+u2 <=0.01)

gamsoutput = 'std';

p.name = 'p';

p.val = num2str(setsize);

ci.name = 'ci';

ci.val = num2str(vertexsize);

c.name = 'c';

c.val = Cost;

c.labels = cellstr(num2str([1:setsize]'));

c.labels = cellstr(num2str([1:setsize]'));

A.name = 'A';

A.val=TechnologyCoefficientMatrix;

A.labels = cellstr(num2str([1:setsize]')) cellstr(num2str([1:vertexsize]'));

[x,z,u1,u2,s]=gams('setpartitioningtrace',c,ci,nt,A,p);

Pay=[0 u1'];

z

for i=1:vertexsize+1

for j=1:vertexsize+1

if i==j

Cbar(i,j)=FakeDistance(i,j);

```

```

else
Cbar(i,j)=FakeDistance(i,j)-Pay(1,i)/2-Pay(1,j)/2;
end
end
end

C(1,:)=Cbar(1,:);
P(1,:)=ones(1,vertexsize+1);
P(1,1)=0;
Load=Demand0;

for v=1:vertexsize-1

clear C1;

for i=2:vertexsize+1

for j=1:vertexsize+1

C1(i,j)=C(v,i)+Cbar(i,j);

cycle=0;

cyclic=0;

if Load(v,i)+Demand0(1,j)> capacity

C1(i,j)=1000000;

else

m=i;

r=1;

while (r < =v)*(1-cyclic)

(1-cyclic);

if P(v-r+1,m)==j

```

```

cycle=cycle+1;

end

if j==1

cyclic=(cycle > 1);

else

cyclic=(cycle >0);

end

m=P(v-r+1,m);

r=r+1;

end

if cyclic>0

C1(i,j)=1000000;

end

end

end

end

end

C2=C1(2:vertexsize+1,:);

[C(v+1,:) P(v+1,.)]=min(C2);

P(v+1,.)=P(v+1,.)+1;

for i=1:vertexsize+1

Load(v+1,i)=Load(v,P(v+1,i))+Demand0(1,i);

end

end

[minpath minpath1]=min(C(:,1));

```



```

if minpath+u2<=0.01
vertex=P(minpath1,1);
newpath=[vertex 1];
pathincomplete=1;
g=1;
while vertex>1
vertex=P(minpath1-g,vertex);
newpath=[vertex newpath];
g=g+1;
end
[newpathsize1 newpathsize]=size(newpath);
sizedifference=setsize1-newpathsize;
if sizedifference >0
for i=1:abs(sizedifference)
newpath=[newpath 0];
end
end
if Pathset(setsize,')==newpath
loop=1;
else
if sizedifference <0
for i=1:abs(sizedifference)
Pathset=[Pathset zeros(setsize,1)];
end

```

```

end

Pathset=[Pathset
newpath];

end

[setsize, setsize1] = size(Pathset);

i=setsize;

Cost(i,1)=0;

for j=1:setsize1-1

if Pathset(i,j+1)> 0

Cost(i)=Cost(i)+FakeDistance(Pathset(i,j),Pathset(i,j+1));

end

end

j=2;

while 1-(Pathset(i,j)==1)

TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;

j=j+1; end

end

iteration=iteration+1;

end

SP(solve).x=x;

infeasibility=0;

if z>=100000

infeasibility=1;

end

```

```

CostR=zeros(setsize,1);

for i=1:setsize

for j=1:setsize-1

if Pathset(i,j+1)>0

CostR(i)=CostR(i)+Distance(Pathset(i,j),Pathset(i,j+1));

end

end

end

z=sum((x')*CostR);

SP(solve).z=z;

integer=1;

i=1;

while integer*(i<setsize)

integer=integer*((x(i)==1)+(x(i)==0));

i=i+1;

end

if (integer==1)*(z<UB)

UB=z;

optimal=x;

optimalset=Pathset;

end

SP(solve).UB=UB;

SP(solve).pathset=Pathset;

SP(solve).iteration=iteration;

```

```

if (1-(integer==1))*(1-infeasibility)*(z<UB)

true=1;

k=1;

q=1;

while true

if x(q,k)==((x(i)==1)+(x(i)==0))

q=q+1;

else

true2=1;

while true2

if SP(solve).branchingidentifier(Pathset(q,k),Pathset(q,k+1))==1

true2=1;

k=k+1;

true=1;

else

branchingindex=[Pathset(q,k) Pathset(q,k+1)];

branchingidentifier1=Pathset(q,k);

branchingidentifier2=Pathset(q,k+1);

true2=0;

true=0;

end

if Pathset(q,k+1)==1

true2=0;

true=1;

```

```

end

end

q=q+1;

end

k=1;

end

branch=branch+1;

SP(branch).oneelements=SP(solve).oneelements;

SP(branch).oneidentifier=SP(solve).oneidentifier;

SP(branch).zeroelements=[SP(solve).zeroelements
branchingindex];

SP(branch).zeroidentifier=SP(solve).zeroidentifier;

SP(branch).zeroidentifier(branchingindex,1)=1;

SP(branch).branchingidentifier=SP(solve).branchingidentifier;

SP(branch).branchingidentifier(branchingidentifier1,branchingidentifier2)=1;

SP(branch).branchingidentifier(branchingidentifier2,branchingidentifier1)=1;

branch=branch+1;

SP(branch).zeroelements=SP(solve).zeroelements;

SP(branch).zeroidentifier=SP(solve).zeroidentifier;

SP(branch).oneelements=[SP(solve).oneelements
branchingindex];

SP(branch).oneidentifier=SP(solve).oneidentifier;

SP(branch).oneidentifier(branchingindex,1)=1;

SP(branch).branchingidentifier=SP(solve).branchingidentifier;

```

```

SP(branch).branchingidentifier(branchingidentifier1,branchingidentifier2)=1;
SP(branch).branchingidentifier(branchingidentifier2,branchingidentifier1)=1;
else
if integer
SP(solve).fathom=1;
fathom=fathom+1;
end
if infeasibility
SP(solve).fathom=2;
fathom=fathom+1;
end
if z>=UB
SP(solve).fathom=3;
fathom=fathom+1;
end
end
solution=[ ];
[optsz optsize1]=size(Pathset);
for i=1:setsz
if x(i,1)>0
solution=[solution
Pathset(i,:)];
end
end
end

```

```
SP(solve).solution=solution;
```

```
SP(solve).s=s;
```

```
solve=solve+1;
```

```
UB
```

```
end CPU TIME=cputime-t
```

APPENDIX D

The Gams Model Of The Set Covering Problem For The Exact Algorithm

```
$ setglobal p 2
$ setglobal ci 8
$ if exist matglobs.gms $ include matglobs.gms
sets i paths/ 1*p/
j city / 1*ci/;

Parameters

c(i) /1 115.2 2 104/;

Table A(i,j) Technology table
      1  2  3  4  5  6
1  0  1  1  0  1  0
2  0  1  0  1  0  1
;

scalar nt /2/;
```



```

variable z;

positive variable x(i);

equation

obj

cover(j)

truck;

obj.. z=e=sum(i,x(i)* c(i));

cover(j) .. sum(i, x(i)*A(i,j)) =e= 1 ;

truck .. sum(i, x(i)) =l= nt ;

MODEL setpartitioningtrace /ALL/;

$ if exist matdata.gms $ include matdata.gms

SOLVE setpartitioningtrace MINIMIZING z USING LP;

display c,A;

set stat /modelstat,solvestat/;

parameter returnStat(stat);

returnStat('modelstat') = setpartitioningtrace.modelstat;

returnStat('solvestat') = setpartitioningtrace.solvestat;

$ libinclude matout x.l I

$ libinclude matout z.l

$ libinclude matout cover.m J

$ libinclude matout truck.m

$ libinclude matout returnStat stat

```

APPENDIX E

The Source Code Of Generalized Petal Heuristic

```
clear all
```

We can start this first step where we sequence the customers using simple TSP heuristics.

```
format long;
```

```
initialization of system parameters
```

```
X=[30 40 37 52 49 49 52 64 20 26 40 30 21 47 17 63 31 62 52 33 51 21 42 41  
31 32 5 25 12 42 36 16 52 41 27 23 17 33 13 13 57 58 62 42 42 57 16 57 8 52 7 38  
27 68 30 48 43 67 58 48 58 27 37 69 38 46 46 10 61 33 62 63 63 69 32 22 45 35 59  
15 5 6 10 17 21 10 5 64 30 15 39 10 32 39 25 32 25 55 48 28 56 37];
```

```
Euclidean distances [x1 x2]=size(X);
```

```
for i=1:x1
```

```
for j=1:x1
```

```
if i==j
```

```
Distance(i,j)=1000000;
```

```

else
Distance(i,j)=round(sqrt((X(i,1)-X(j, 1))^2+(X(i,2)-X(j, 2))^2));
end
end
end

Demand=[7 30 16 9 21 15 19 23 11 5 19 29 23 21 10 15 3 41 9 28 8 8 16 10 28
7 15 14 6 19 11 12 23 26 17 6 9 15 14 7 27 13 11 16 10 5 25 17 18 10];

capacity=160;

nt=7;

t=cputime;

Demand2=[Demand Demand];

[vertexsize,v1] = size(Distance);

vertexsize=vertexsize-1;

Nearest Neighborhood Heuristic

for h=1:vertexsize

currentptnnh=h;

Distancennh=Distance(2:vertexsize+1,2:vertexsize+1);

Distancennh(:,currentptnnh)=1000000*ones(vertexsize,1);

Sequencedvertexset=[currentptnnh];

for i=1:vertexsize-1;

[min1 min2]=min(Distancennh(currentptnnh,:));

currentptnnh=min2;

Sequencedvertexset=[Sequencedvertexset currentptnnh];

Distancennh(:,currentptnnh)=1000000*ones(vertexsize,1);

```

```

end

tsp=[Sequencedvertexset(1,1:vertexsize) Sequencedvertexset(1,1)];

improvement=1;

while improvement

for i=1:vertexsize

sequence(tsp(1,i))=i;

end

for i=1:vertexsize

for j=i+1:vertexsize

if (sequence(1,i)-sequence(1,j)>1)+ (sequence(1,i)- sequence(1,j)<-
1)+(sequence(1,i)- sequence(1,j)<9)+(sequence(1,i)-sequence(1,j)>-9)

swapgain(i,j)=Distance(i+1,tsp(sequence(1,i)+1)+1)+
Distance(j+1,tsp(sequence(1,j)+1)+1)- Distance(i+1,j+1)-
Distance(tsp(sequence(1,i)+1)+1,tsp(sequence(1,j)+1)+1);

end

end

end

[z11 z1]=max(swapgain);

[z22 z2]=max(max(swapgain));

z1=z1(1,z2);

if z22>0.001

if sequence(1,z1)>sequence(1,z2)

z=z2;

z2=z1;


```

```

z1=z;

end

newpath=[tsp(1,1:sequence(1,z1)) z2 ];

for i=1:(sequence(1,z2)-sequence(1,z1)-1)

newpath=[newpath tsp(1,sequence(1,z2)-i)];

end

newpath=[newpath tsp(1,sequence(1,z2)+1:vertexsize+1)];

tsp=newpath;

else

improvement=0;

end

end

Sequencedvertexset=tsp(1,1:vertexsize);

Sequencedvertexset=[Sequencedvertexset Sequencedvertexset];

iter=1;

for i=1:vertexsize

true=1;

set=[ ];

setcapacity=0;

j=0;

while true

if setcapacity + Demand2(1,Sequencedvertexset(1,i+j))<=capacity

if j==0

Petal(iter,1)=Sequencedvertexset(1,i+j);

```

```

Pathset(iter,1)=Sequencedvertexset(1,i+j)+1;

setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));

iter=iter+1;

j=j+1;

else

if j>0

Petal(iter,:)=Petal(iter-1,:);

Pathset(iter,:)=Pathset(iter-1,:);

Petal(iter,j+1)=Sequencedvertexset(1,i+j);

Pathset(iter,j+1)=Sequencedvertexset(1,i+j)+1;

setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));

iter=iter+1;

j=j+1;

end

end

else

true=0;

end

end

end

[Petalsize1 Petalsize2]=size(Petal);

Pathset=[ones(Petalsize1,1) Pathset];

[Pathsetsize1 Pathsetsize2]=size(Pathset);

for i=1:Petalsize1

```

```

[min1 min2]=min(Pathset(i,:));
if min1==0
Pathset(i,min2)=1;
else
Pathset(i,Pathsetsize2+1)=1;
Pathsetsize2=Pathsetsize2+1;
end
end

Cost=zeros(Pathsetsize1,1);
for i=1:Pathsetsize1
for j=1:Pathsetsize2-1
if Pathset(i,j+1)>0
Cost(i)=Cost(i)+Distance(Pathset(i,j),Pathset(i,j+1));
end
end
end

for i=1:Pathsetsize1
j=2;
while 1-(Pathset(i,j)==1)
TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;
j=j+1;
end
end

gamsoutput = 'std';

```

```

p.name = 'p';

p.val = num2str(Pathsetsize1);

ci.name = 'ci';

ci.val = num2str(vertexsize);

c.name = 'c';

c.val = Cost;

c.labels = cellstr(num2str([1:Pathsetsize1]'));

A.name = 'A';

A.val=TechnologyCoefficientMatrix;

A.labels          =          cellstr(num2str([1:Pathsetsize1]'))
cellstr(num2str([1:vertexsize]'));

[x,z,u1,u2,s]=gams('petalpartitioning',c,ci,nt,A,p);

solution=[ ];

[optsz optsz1]=size(Pathset);

for i=1:Pathsetsize1

if x(i,1)>0

solution=[solution
Pathset(i,:)];

end

end

[so1 so2]=size(solution);

Load=zeros(so1,1);

for i=1:so1

for j=2:so2-1

```



```

if solution(i,j+1)>0
Load(i)=Load(i)+Demand(solution(i,j)-1);
end
end
end

for i=1:s01
for j=2:s02
if solution(i,j)>1
dim1(solution(i,j))=i;
dim2(solution(i,j))=j;
end
end
end

m=1;
while m>0.01
iter=0;
for i=2:vertexsize
for j=i+1:vertexsize+1
if (Load(dim1(i))-Demand(i-1)+Demand(j-1)<capacity)*(Load(dim1(j))-
Demand(j-1)+Demand(i-1)<capacity)
exchangegain(i,j)=Distance(i,solution(dim1(i),dim2(i)+1))+
Distance(i,solution(dim1(i),dim2(i)-1))+
Distance(j,solution(dim1(j),dim2(j)+1))+ Distance(j,solution(dim1(j),dim2(j)-
1))-
Distance(j,solution(dim1(i),dim2(i)+1))-

```

```

Distance(j,solution(dim1(i),dim2(i)-1))-
Distance(i,solution(dim1(j),dim2(j)+1))-
Distance(i,solution(dim1(j),dim2(j)-1));

    end

    end

    end

[m m1]=max(max(exchangegain'));
[m m2]=max(max(exchangegain));
if m>0
    solution(dim1(m1),dim2(m1))=m2;
    solution(dim1(m2),dim2(m2))=m1;
    mm2=dim2(m2);
    mm1=dim1(m2);
    dim1(m2)=dim1(m1);
    dim2(m2)=dim2(m1);
    dim1(m1)=mm1;
    dim2(m1)=mm2;
    z=z-exchangegain(m1,m2);
    exchangegain=zeros(vertexsize);
end
end
HS(1,h).solution=solution;
HS(1,h).z=z;
clear Petal;

```

```

clear Pathset;

clear Cost;

clear TechnologyCoefficientMatrix;

h

z

end

Nearest Insertion Heuristic for h=1:vertexsize

i=1;

currentptnih=h;

Distancenih=Distance(2:vertexsize+1,2:vertexsize+1);

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent=[Distancenih(currentptnih,:)];

Sequencedvertexset=[currentptnih];

[min1 min2]=min(Distancenihcurrent);

currentptnih=min2;

Sequencedvertexset=[Sequencedvertexset currentptnih];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent

Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

for i=2:vertexsize-1;

[min1 min2]=min(min(Distancenihcurrent));

currentptnih=min2;

```

```

clear insertioncost;

insertioncost(1,1)=-Distance(Sequencedvertexset(1,1)+1,1)+
Distance(currentptnih+1,Sequencedvertexset(1,1)+1)+
Distance(currentptnih+1,1);

for j=2:s1
    insertioncost(1,j)=-Distance(Sequencedvertexset(1,j-
1)+1,Sequencedvertexset(1,j)+1)+
Distance(currentptnih+1,Sequencedvertexset(1,j-1)+1)+
Distance(currentptnih+1,Sequencedvertexset(1,j)+1);
end

insertioncost(1,s1+1)=-Distance(Sequencedvertexset(1,s1)+1,1)+
Distance(currentptnih+1,Sequencedvertexset(1,s1)+1)+
Distance(currentptnih+1,1);

[z1 z2]=min(insertioncost);

Sequencedvertexset=[Sequencedvertexset(1,1:z2-1)           currentptnih
Sequencedvertexset(1,z2:s1)];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent
Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

end

tsp=[Sequencedvertexset(1,1:vertexsize) Sequencedvertexset(1,1)];

improvement=1;

```

```

while improvement
for i=1:vertexsize
sequence(tsp(1,i))=i;
end

for i=1:vertexsize
for j=i+1:vertexsize
if (sequence(1,i)-sequence(1,j)>1)+ (sequence(1,i)-sequence(1,j)<-1)+
(sequence(1,i)-sequence(1,j)<9)+ (sequence(1,i)-sequence(1,j)>-9)
swapgain(i,j)=Distance(i+1,tsp(sequence(1,i)+1)+1)+
Distance(j+1,tsp(sequence(1,j)+1)+1)- Distance(i+1,j+1)-
Distance(tsp(sequence(1,i)+1)+1,tsp(sequence(1,j)+1)+1);
end
end
end

[z11 z1]=max(swapgain);
[z22 z2]=max(max(swapgain));
z1=z1(1,z2);
if z22>0.001
if sequence(1,z1)>sequence(1,z2)
z=z2;
z2=z1;
z1=z;
end
newpath=[tsp(1,1:sequence(1,z1)) z2 ];

```

```

for i=1:(sequence(1,z2)-sequence(1,z1)-1)
newpath=[newpath tsp(1,sequence(1,z2)-i)];
end
newpath=[newpath tsp(1,sequence(1,z2)+1:vertexsize+1)];
tsp=newpath;
else
improvement=0;
end
end
end
Sequencedvertexset=tsp(1,1:vertexsize);
Sequencedvertexset=[Sequencedvertexset Sequencedvertexset];
iter=1;
for i=1:vertexsize
true=1;
setcapacity=0;
j=0;
while true
if setcapacity + Demand2(1,Sequencedvertexset(1,i+j))<=capacity
if j==0
Petal(iter,1)=Sequencedvertexset(1,i+j);
Pathset(iter,1)=Sequencedvertexset(1,i+j)+1;
setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));
iter=iter+1;
j=j+1;

```

```

else
if j>0
Petal(iter,:)=Petal(iter-1,:);
Pathset(iter,:)=Pathset(iter-1,:);
Petal(iter,j+1)=Sequencedvertexset(1,i+j);
Pathset(iter,j+1)=Sequencedvertexset(1,i+j)+1;
setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));
iter=iter+1;
j=j+1;
end
end
else
true=0;
end
end
end

[Petalsize1 Petalsize2]=size(Petal);
Pathset=[ones(Petalsize1,1) Pathset];
[Pathsetsize1 Pathsetsize2]=size(Pathset);
for i=1:Petalsize1
[min1 min2]=min(Pathset(i,:));
if min1==0
Pathset(i,min2)=1;
else

```

```

Pathset(i,Pathsetsize2+1)=1;
Pathsetsize2=Pathsetsize2+1;
end
end
Cost=zeros(Pathsetsize1,1);
for i=1:Pathsetsize1
for j=1:Pathsetsize2-1
if Pathset(i,j+1)>0
Cost(i)=Cost(i)+Distance(Pathset(i,j),Pathset(i,j+1));
end end
end
for i=1:Pathsetsize1
j=2;
while 1-(Pathset(i,j)==1)
TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;
j=j+1;
end
end
gamsoutput = 'std';
p.name = 'p';
p.val = num2str(Pathsetsize1);
ci.name = 'ci';
ci.val = num2str(vertexsize);
c.name = 'c';

```



```

c.val = Cost;

c.labels = cellstr(num2str([1:Pathsetsize1]'));

A.name = 'A';

A.val=TechnologyCoefficientMatrix;

A.labels          =          cellstr(num2str([1:Pathsetsize1]'))
cellstr(num2str([1:vertexsize]'));

[x,z,u1,u2,s]=gams('petalpartitioning',c,ci,nt,A,p);

solution=[ ];

[optsz optsz1]=size(Pathset);

for i=1:Pathsetsize1

if x(i,1)>0

solution=[solution
Pathset(i,:)];

end

end

[so1 so2]=size(solution);

Load=zeros(so1,1);

for i=1:so1

for j=2:so2-1

if solution(i,j+1)>0

Load(i)=Load(i)+Demand(solution(i,j)-1);

end

end

end

```

```

for i=1:so1
for j=2:so2
if solution(i,j)>1
dim1(solution(i,j))=i;
dim2(solution(i,j))=j;
end
end
end

m=1;
while m>0.01
for i=2:vertexsize
for j=i+1:vertexsize+1
if (Load(dim1(i))-Demand(i-1)+Demand(j-1)<capacity)*(Load(dim1(j))-
Demand(j-1)+Demand(i-1)<capacity)
exchangegain(i,j)=Distance(i,solution(dim1(i),dim2(i)+1))+
Distance(i,solution(dim1(i),dim2(i)-1))+
Distance(j,solution(dim1(j),dim2(j)+1))+ Distance(j,solution(dim1(j),dim2(j)-
1))-
Distance(j,solution(dim1(i),dim2(i)+1))-
Distance(j,solution(dim1(i),dim2(i)-1))-
Distance(i,solution(dim1(j),dim2(j)+1))-
Distance(i,solution(dim1(j),dim2(j)-1));
end
end
end

```

```

[m m1]=max(max(exchangegain'));
[m m2]=max(max(exchangegain));
if m>0
solution(dim1(m1),dim2(m1))=m2;
solution(dim1(m2),dim2(m2))=m1;
mm2=dim2(m2);
mm1=dim1(m2);
dim1(m2)=dim1(m1);
dim2(m2)=dim2(m1);
dim1(m1)=mm1;
dim2(m1)=mm2;
z=z-exchangegain(m1,m2);
exchangegain=zeros(vertexsize);
end
end
HS(2,h).solution=solution;
HS(2,h).z=z;
clear Petal;
clear Pathset;
clear Cost;
clear TechnologyCoefficientMatrix;
h
z
end

```

```

Farthest Insertion Heuristic

for h=1:vertexsize

i=1;

Notincludedset=ones(1,vertexsize);

currentptnih=h;

Notincludedset(1,currentptnih)=0;

Distancenih=Distance(2:vertexsize+1,2:vertexsize+1);

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent=[Distancenih(currentptnih,:)];

Sequencedvertexset=[currentptnih];

Notincludedsetdummy=Notincludedset;

clear insertion;

for k=1:vertexsize-1

[currentptnih1 currentptnih]=max(Notincludedsetdummy);

insertion(k,1)=-Distance(Sequencedvertexset(1,1)+1,1)+

Distance(currentptnih+1,Sequencedvertexset(1,1)+1)+

Distance(currentptnih+1,1);

Notincludedsetdummy(1,currentptnih)=0;

end

[z11 z1]=min(insertion);

u=0;

sum=0;

while sum<z1

u=u+1;

```

```

sum=sum+Notincludedset(1,u);

end

currentptnih=u;

Notincludedset(1,currentptnih)=0;

Sequencedvertexset=[Sequencedvertexset currentptnih];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent

Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

for i=2:vertexsize-1;

Notincludedsetdummy=Notincludedset;

clear insertioncost;

for k=1:vertexsize-s1

[currentptnih1 currentptnih]=max(Notincludedsetdummy);

insertioncost(k,1)=-Distance(Sequencedvertexset(1,1)+1,1)+

Distance(currentptnih+1,Sequencedvertexset(1,1)+1)+

Distance(currentptnih+1,1);

for j=2:s1

insertioncost(k,j)=-Distance(Sequencedvertexset(1,j-

1)+1,Sequencedvertexset(1,j)+1)+

Distance(currentptnih+1,Sequencedvertexset(1,j-1)+1)+

Distance(currentptnih+1,Sequencedvertexset(1,j)+1);

end

```

```

insertioncost(k,s1+1)=-Distance(Sequencedvertexset(1,s1)+1,1)+
Distance(currentptnih+1,Sequencedvertexset(1,s1)+1)+
Distance(currentptnih+1,1);

Notincludedsetdummy(1,currentptnih)=0;

end

[zz1 z2]=min(insertioncost');

[z22 z1]=max(min(insertioncost'));

z2=z2(1,z1);

u=0;

sum=0;

while sum<z1 u=u+1;

sum=sum+Notincludedset(1,u);

end

currentptnih=u;

Sequencedvertexset=[Sequencedvertexset(1,1:z2-1)           currentptnih
Sequencedvertexset(1,z2:s1)];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent
Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

Notincludedset(1,currentptnih)=0;

end

tsp=[Sequencedvertexset(1,1:vertexsize) Sequencedvertexset(1,1)];

```

```

improvement=1;

while improvement

for i=1:vertexsize

sequence(tsp(1,i))=i;

end

for i=1:vertexsize

for j=i+1:vertexsize

if (sequence(1,i)-sequence(1,j)>1)+( sequence(1,i)- sequence(1,j)<-1)+
(sequence(1,i)- sequence(1,j)<9)+ (sequence(1,i)- sequence(1,j)>-9)

swapgain(i,j)=

Distance(i+1,tsp(sequence(1,i)+1)+1)+ Distance(j+1,tsp(sequence(1,j)+1)+1)-
Distance(i+1,j+1)- Distance(tsp(sequence(1,i)+1)+1,tsp(sequence(1,j)+1)+1);

end

end

end

[z11 z1]=max(swapgain);

[z22 z2]=max(max(swapgain));

z1=z1(1,z2);

if z22>0.001

if sequence(1,z1)>sequence(1,z2)

z=z2;

z2=z1;

z1=z;

end

```

```

newpath=[tsp(1,1:sequence(1,z1)) z2 ];
for i=1:(sequence(1,z2)-sequence(1,z1)-1)
newpath=[newpath tsp(1,sequence(1,z2)-i)];
end
newpath=[newpath tsp(1,sequence(1,z2)+1:vertexsize+1)];
tsp=newpath;
else
improvement=0;
end
end
end
Sequencedvertexset=tsp(1,1:vertexsize);
Sequencedvertexset=[Sequencedvertexset Sequencedvertexset];
iter=1;
for i=1:vertexsize
true=1;
set=[ ];
setcapacity=0;
j=0;
while true if setcapacity + Demand2(1,Sequencedvertexset(1,i+j))<=capacity
if j==0
Petal(iter,1)=Sequencedvertexset(1,i+j);
Pathset(iter,1)=Sequencedvertexset(1,i+j)+1;
setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));
iter=iter+1;

```



```

j=j+1;

else

if j>0

Petal(iter,:)=Petal(iter-1,:);

Pathset(iter,:)=Pathset(iter-1,:);

Petal(iter,j+1)=Sequencedvertexset(1,i+j);

Pathset(iter,j+1)=Sequencedvertexset(1,i+j)+1;

setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));

iter=iter+1;

j=j+1;

end

end

else

true=0;

end

end

end [Petalsize1 Petalsize2]=size(Petal);

Pathset=[ones(Petalsize1,1) Pathset];

[Pathsetsize1 Pathsetsize2]=size(Pathset);

for i=1:Petalsize1

[min1 min2]=min(Pathset(i,:));

if min1==0

Pathset(i,min2)=1;

else

```

```

Pathset(i,Pathsetsize2+1)=1;
Pathsetsize2=Pathsetsize2+1;
end
end
Cost=zeros(Pathsetsize1,1);
for i=1:Pathsetsize1
for j=1:Pathsetsize2-1
if Pathset(i,j+1)>0
Cost(i)=Cost(i)+Distance(Pathset(i,j),Pathset(i,j+1));
end
end
end
for i=1:Pathsetsize1
j=2;
while 1-(Pathset(i,j)==1)
TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;
j=j+1;
end
end
gamsoutput = 'std';
p.name = 'p';
p.val = num2str(Pathsetsize1);
ci.name = 'ci';
ci.val = num2str(vertexsize);

```

```

c.name = 'c';

c.val = Cost;

c.labels = cellstr(num2str([1:Pathsetsize1]'));

A.name = 'A';

A.val=TechnologyCoefficientMatrix;

A.labels          =          cellstr(num2str([1:Pathsetsize1]'))

cellstr(num2str([1:vertexsize]'));

[x,z,u1,u2,s]=gams('petalpartitioning',c,ci,nt,A,p);

solution=[ ];

[optsz optsz1]=size(Pathset);

for i=1:Pathsetsize1

if x(i,1)>0

solution=[solution

Pathset(i,:)];

end

end

[so1 so2]=size(solution);

Load=zeros(so1,1);

for i=1:so1

for j=2:so2-1

if solution(i,j+1)>0

Load(i)=Load(i)+Demand(solution(i,j)-1);

end

end

```

```

end

for i=1:so1

for j=2:so2

if solution(i,j)>1

dim1(solution(i,j))=i;

dim2(solution(i,j))=j; end

end

end

m=1;

while m>0.01

for i=2:vertexsize

for j=i+1:vertexsize+1

if (Load(dim1(i))-Demand(i-1)+Demand(j-1)<capacity)*(Load(dim1(j))-
Demand(j-1)+Demand(i-1)<capacity)

exchangegain(i,j)= Distance(i,solution(dim1(i),dim2(i)+1))+
Distance(i,solution(dim1(i),dim2(i)-1))+
Distance(j,solution(dim1(j),dim2(j)+1))+ Distance(j,solution(dim1(j),dim2(j)-
1))-
Distance(j,solution(dim1(i),dim2(i)+1))-
Distance(j,solution(dim1(i),dim2(i)-1))-
Distance(i,solution(dim1(j),dim2(j)+1))-
Distance(i,solution(dim1(j),dim2(j)-1));

end

end

end

```

```

[m m1]=max(max(exchangeGain'));
[m m2]=max(max(exchangeGain));
if m>0
solution(dim1(m1),dim2(m1))=m2;
solution(dim1(m2),dim2(m2))=m1;
mm2=dim2(m2);
mm1=dim1(m2);
dim1(m2)=dim1(m1);
dim2(m2)=dim2(m1);
dim1(m1)=mm1;
dim2(m1)=mm2;
z=z-exchangeGain(m1,m2);
exchangeGain=zeros(vertexSize);
end
end
HS(3,h).solution=solution;
HS(3,h).z=z;
clear Petal;
clear Pathset;
clear Cost;
clear TechnologyCoefficientMatrix;
h
z
end

```

Cheapest Insertion Heuristics

```
for h=1:vertexsize
    i=1;
    Notincludedset=ones(1,vertexsize);
    currentptnih=h;
    Notincludedset(1,currentptnih)=0;
    Distancenih=Distance(2:vertexsize+1,2:vertexsize+1);
    Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);
    Distancenihcurrent=[Distancenih(currentptnih,:)];
    Sequencedvertexset=[currentptnih];
    Notincludedsetdummy=Notincludedset;
    clear insertion;
    for k=1:vertexsize-1
        [currentptnih1 currentptnih]=max(Notincludedsetdummy);
        insertion(k,1)=-Distance(Sequencedvertexset(1,1)+1,1)+
Distance(currentptnih+1,Sequencedvertexset(1,1)+1)+
Distance(currentptnih+1,1);
        Notincludedsetdummy(1,currentptnih)=0;
    end
    [z11 z1]=min(insertion);
    u=0;
    sum=0;
    while sum<z1
        u=u+1;
```

```

sum=sum+Notincludedset(1,u);

end

currentptnih=u;

Notincludedset(1,currentptnih)=0;

Sequencedvertexset=[Sequencedvertexset currentptnih];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent

Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

for i=2:vertexsize-1;

Notincludedsetdummy=Notincludedset;

clear insertioncost;

for k=1:vertexsize-s1

[currentptnih1 currentptnih]=max(Notincludedsetdummy);

insertioncost(k,1)=-Distance(Sequencedvertexset(1,1)+1,1)+

Distance(currentptnih+1,Sequencedvertexset(1,1)+1)+

Distance(currentptnih+1,1);

for j=2:s1

insertioncost(k,j)=-Distance(Sequencedvertexset(1,j-

1)+1,Sequencedvertexset(1,j)+1)+

Distance(currentptnih+1,Sequencedvertexset(1,j-1)+1)+

Distance(currentptnih+1,Sequencedvertexset(1,j)+1);

end

```

```

insertioncost(k,s1+1)=-Distance(Sequencedvertexset(1,s1)+1,1)+
Distance(currentptnih+1,Sequencedvertexset(1,s1)+1)+
Distance(currentptnih+1,1);

Notincludedsetdummy(1,currentptnih)=0;

end

[z11 z1]=min(min(insertioncost'));
[z22 z2]=min(min(insertioncost));

u=0;

sum=0;

while sum<z1

u=u+1;

sum=sum+Notincludedset(1,u);

end

currentptnih=u;

Sequencedvertexset=[Sequencedvertexset(1,1:z2-1)           currentptnih
Sequencedvertexset(1,z2:s1)];

Distancenih(:,currentptnih)=1000000*ones(vertexsize,1);

Distancenihcurrent(:,currentptnih)=1000000*ones(i,1);

Distancenihcurrent=[Distancenihcurrent

Distancenih(currentptnih,:)];

[s1 s2]=size(Distancenihcurrent);

Notincludedset(1,currentptnih)=0;

end

tsp=[Sequencedvertexset(1,1:vertexsize) Sequencedvertexset(1,1)];

```



```

improvement=1;

while improvement

for i=1:vertexsize

sequence(tsp(1,i))=i;

end

for i=1:vertexsize

for j=i+1:vertexsize

if      (sequence(1,i)-sequence(1,j)>1)+(sequence(1,i)-sequence(1,j)<-
1)+(sequence(1,i)-sequence(1,j)<9)+(sequence(1,i)-sequence(1,j)>-9)

    swapgain(i,j)=
Distance(i+1,tsp(sequence(1,i)+1)+1) + Distance(j+1,tsp(sequence(1,j)+1)+1)
- Distance(i+1,j+1) - Distance(tsp(sequence(1,i)+1)+1,tsp(sequence(1,j)+1)+1);

end

end

end

[z11 z1]=max(swapgain);

[z22 z2]=max(max(swapgain));

z1=z1(1,z2);

if z22>0.001

if sequence(1,z1)>sequence(1,z2)

z=z2;

z2=z1;

z1=z;

end

```

```

newpath=[tsp(1,1:sequence(1,z1)) z2 ];
for i=1:(sequence(1,z2)-sequence(1,z1)-1)
newpath=[newpath tsp(1,sequence(1,z2)-i)];
end
newpath=[newpath tsp(1,sequence(1,z2)+1:vertexsize+1)];
tsp=newpath;
else improvement=0;
end
end
end
Sequencedvertexset=tsp(1,1:vertexsize);
Sequencedvertexset=[Sequencedvertexset Sequencedvertexset];
iter=1;
for i=1:vertexsize
true=1;
set=[ ];
setcapacity=0;
j=0;
while true
if setcapacity + Demand2(1,Sequencedvertexset(1,i+j))<=capacity
if j==0
Petal(iter,1)=Sequencedvertexset(1,i+j);
Pathset(iter,1)=Sequencedvertexset(1,i+j)+1;
setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));
iter=iter+1;

```

```

j=j+1;

else

if j>0

Petal(iter,:)=Petal(iter-1,:);

Pathset(iter,:)=Pathset(iter-1,:);

Petal(iter,j+1)=Sequencedvertexset(1,i+j);

Pathset(iter,j+1)=Sequencedvertexset(1,i+j)+1;

setcapacity=setcapacity+Demand2(1,Sequencedvertexset(1,i+j));

iter=iter+1;

j=j+1;

end

end

else

true=0;

end

end

end

[Petalsize1 Petalsize2]=size(Petal);

Pathset=[ones(Petalsize1,1) Pathset];

[Pathsetsize1 Pathsetsize2]=size(Pathset);

for i=1:Petalsize1

[min1 min2]=min(Pathset(i,:));

if min1==0

Pathset(i,min2)=1;

```

```

else
Pathset(i,Pathsetsize2+1)=1;
Pathsetsize2=Pathsetsize2+1;
end
end

Cost=zeros(Pathsetsize1,1);
for i=1:Pathsetsize1
for j=1:Pathsetsize2-1
if Pathset(i,j+1)>0
Cost(i)=Cost(i)+Distance(Pathset(i,j),Pathset(i,j+1));
end
end
end

for i=1:Pathsetsize1
j=2;
while 1-(Pathset(i,j)==1)
TechnologyCoefficientMatrix(i,Pathset(i,j)-1)=1;
j=j+1;
end
end

gamsoutput = 'std';

p.name = 'p';

p.val = num2str(Pathsetsize1);

ci.name = 'ci';

```

```

ci.val = num2str(vertexsize);

c.name = 'c';

c.val = Cost;

c.labels = cellstr(num2str([1:Pathsetsize1]'));

A.name = 'A';

A.val=TechnologyCoefficientMatrix;

A.labels          =          cellstr(num2str([1:Pathsetsize1]'))
cellstr(num2str([1:vertexsize]'));

[x,z,u1,u2,s]=gams('petalpartitioning',c,ci,nt,A,p);

solution=[ ];

[optsz optsz1]=size(Pathset);

for i=1:Pathsetsize1

if x(i,1)>0

solution=[solution Pathset(i,:)];

end

end

[so1 so2]=size(solution);

Load=zeros(so1,1);

for i=1:so1

for j=2:so2-1

if solution(i,j+1)>0

Load(i)=Load(i)+Demand(solution(i,j)-1);

end

end

```

```

end

for i=1:so1

for j=2:so2

if solution(i,j)>1

dim1(solution(i,j))=i;

dim2(solution(i,j))=j;

end

end

end

m=1;

while m>0.01

for i=2:vertexsize

for j=i+1:vertexsize+1

if (Load(dim1(i))-Demand(i-1)+Demand(j-1)<capacity)*(Load(dim1(j))-
Demand(j-1)+Demand(i-1)<capacity)

exchangegain(i,j)=Distance(i,solution(dim1(i),dim2(i)+1))+
Distance(i,solution(dim1(i),dim2(i)-1))+
Distance(j,solution(dim1(j),dim2(j)+1))+
Distance(j,solution(dim1(j),dim2(j)-1))-Distance(j,solution(dim1(i),dim2(i)+1))-
Distance(j,solution(dim1(i),dim2(i)-1))-Distance(i,solution(dim1(j),dim2(j)+1))-
Distance(i,solution(dim1(j),dim2(j)-1));

end

end

end

```

```

[m m1]=max(max(exchangegain'));
[m m2]=max(max(exchangegain));
if m>0
solution(dim1(m1),dim2(m1))=m2;
solution(dim1(m2),dim2(m2))=m1;
mm2=dim2(m2);
mm1=dim1(m2);
dim1(m2)=dim1(m1);
dim2(m2)=dim2(m1);
dim1(m1)=mm1;
dim2(m1)=mm2;
z=z-exchangegain(m1,m2);
exchangegain=zeros(vertexsize);
end
end
HS(4,h).solution=solution;
HS(4,h).z=z;
clear Petal;
clear Pathset;
clear Cost;
clear TechnologyCoefficientMatrix;
h
z
end for i=1:4

```

```
for j=1:vertexsize
obj(j,i)=HS(i,j).z
end
end CPU TIME=cputime-t;
```


APPENDIX F

The Gams Model Of The Set Covering Problem For The Generalized Petal Heuristic

```
$ setglobal p 2  
  
$ setglobal ci 10  
  
$ if exist matglobs.gms $ include matglobs.gms  
  
sets i paths/ 1*p/  
j city / 1*ci/  
  
Parameters  
  
c(i) /1 115.2 2 104/ ;  
  
Table A(i,j) Technology table  
      1  2  3  4  5  6  
1  0  1  1  0  1  0  
2  1  1  0  1  0  1  
;  
  
scalar nt /2/;
```

```

variable z;

positive variable x(i);

equation

obj

cover(j)

truck;

obj.. z=e=sum(i,x(i)* c(i));

cover(j) .. sum(i, x(i)*A(i,j)) =e= 1 ;

truck .. sum(i, x(i)) =l= nt ;

MODEL setpartitioningtrace /ALL/;

$ if exist matdata.gms $ include matdata.gms

SOLVE setpartitioningtrace MINIMIZING z Using LP;

display c,A;

set stat /modelstat,solvestat/;

parameter returnStat(stat);

returnStat('modelstat') = setpartitioningtrace.modelstat;

returnStat('solvestat') = setpartitioningtrace.solvestat;

$ libinclude matout x.l I

$ libinclude matout z.l

$ libinclude matout cover.m J

$ libinclude matout truck.m

$ libinclude matout returnStat stat

```

APPENDIX G

The Solution Of The Arranging Algorithm For A Given Data

Variable	Length (mm)	Height (mm)	Width (mm)	Weight (tone)
X1	1074	1241	180	0.121799874
X2	1630	1825	700	0.61248096
X3	1728	1864	700	0.369139433
X4	1830	2770	130	0.44837928
X5	498	936	20	0.009173399
X6	508	1051	96	0.02808092
X7	459	804	80	0.018472356
X8	535	925	120	0.024435003
X9	775	1215	24	0.01853118
X10	377	1107	48	0.010947945
X11	683	2030	22	0.027286123
X12	994	1440	110	0.141719222
X13	445	490	8	0.00426933
X14	390	860	32	0.020471136
X15	970	2990	12	0.051341184
X16	654	1136	648	0.137151475
X17	1127	1997	620	0.333488778
X18	1180	1480	432	0.338074179
X19	679	954	180	0.076671017
X20	799	1034	324	0.120491745
X21	4253	2609	700	1.392962621
Total	21588			4.30536716

APPENDIX H

The Distance Matrix For The Given Data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	M	52	50	47	52	54	54	52	52	52	52	50	52	35	35	32	54	35	52	52	52	35
1	52	M	9.6	8	2.4	3	3	2.4	0	2.4	11	9.6	11	19.5	18	32	3	19.5	0	0	0	19.5
2	50	9.6	M	1.6	10	10.6	10.6	10	9.6	10	1.4	0	1.4	23	21.5	29.1	10.6	23	9.6	9.6	9.6	23
3	47	8	1.6	M	9	9.6	9.6	9	8	9	3	1.6	3	21.4	19.9	27.5	9	21.4	8	8	8	21.4
4	52	2.4	10	9	M	1.6	1.6	0	2.4	0	11.4	10	11.4	21	20	34	1.6	21	2.4	2.4	2.4	21
5	54	3	10.6	9.6	1.6	M	0	1.6	3	1.6	13	11.6	13	22.6	21.6	35.6	0	22.6	3	3	3	22.6
6	54	3	10.6	9.6	1.6	0	M	1.6	3	1.6	13	11.6	13	22.6	21.6	35.6	0	22.6	3	3	3	22.6
7	52	2.4	10	9	0	1.6	1.6	M	2.4	0	11.4	10	11.4	21	20	34	1.6	21	2.4	2.4	2.4	21
8	52	0	9.6	8	2.4	3	3	2.4	M	2.4	11	9.6	11	19.5	18	32	3	19.5	0	0	0	19.5
9	52	2.4	10	9	0	1.6	1.6	0	2.4	M	11.4	10	11.4	21	20	34	1.6	21	2.4	2.4	2.4	21
10	52	11	1.4	3	11.4	13	13	11.4	11	11.4	M	1.4	0	24.4	22.9	31.3	13	24.4	11	11	11	24.4
11	50	9.6	0	1.6	10	11.6	11.6	10	9.6	10	1.4	M	1.4	23	21.5	29.1	10.6	23	9.6	9.6	9.6	23
12	52	11	1.4	3	11.4	13	13	11.4	11	11.4	0	1.4	M	24.4	22.9	31.3	13	24.4	11	11	11	24.4
13	35	19.5	23	21.4	21	22.6	22.6	21	19.5	21	24.4	23	24.4	M	1	15	22.6	0	19.5	19.5	19.5	0
14	35	18	21.5	19.9	20	21.6	21.6	20	18	20	22.9	21.5	22.9	1	M	13.5	21.6	1	18	18	18	13.5
15	32	32	29.1	27.5	34	35.6	35.6	34	32	34	31.3	29.1	31.3	15	13.5	M	35.6	15	32	32	32	15
16	54	3	10.6	9	1.6	0	0	1.6	3	1.6	13	10.6	13	22.6	21.6	35.6	M	22.6	3	3	3	22.6
17	35	19.5	23	21.4	21	22.6	22.6	21	19.5	21	24.4	23	24.4	0	1	15	22.6	M	19.5	19.5	19.5	0
18	52	0	9.6	8	2.4	3	3	2.4	0	2.4	11	9.6	11	19.5	18	32	3	19.5	M	0	0	19.5
19	52	0	9.6	8	2.4	3	3	2.4	0	2.4	11	9.6	11	19.5	18	32	3	19.5	0	M	0	19.5
20	52	0	9.6	8	2.4	3	3	2.4	0	2.4	11	9.6	11	19.5	18	32	3	19.5	0	0	M	19.5
21	35	19.5	23	21.4	21	22.6	22.6	21	19.5	21	24.4	23	24.4	0	13.5	15	22.6	0	19.5	19.5	19.5	M