A VARIABLE NEIGHBORHOOD SEARCH PROCEDURE FOR THE COMBINED LOCATION WITH PARTIAL COVERAGE AND SELECTIVE TRAVELING SALESMAN PROBLEM

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

BY

FATİH RAHİM

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

MAY 2010

Approval of the thesis:

A VARIABLE NEIGHBORHOOD SEARCH PROCEDURE FOR THE COMBINED LOCATION WITH PARTIAL COVERAGE AND SELECTIVE TRAVELING SALESMAN PROBLEM

submitted by **FATİH RAHİM** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Nur Evin Özdemirel _____
Head of Department, **Industrial Engineering**

Assoc. Prof. Dr. Canan Sepil _____
Supervisor, **Industrial Engineering Dept.**, **METU**

Examining Committee Members:

Assoc. Prof. Dr. Haldun Süral _____
Industrial Engineering Dept., METU

Assoc. Prof. Dr. Canan Sepil _____
Industrial Engineering Dept., METU

Assoc. Prof. Dr. Yasemin Serin _____
Industrial Engineering Dept., METU

Assist. Prof. Dr. Sedef Meral _____
Industrial Engineering Dept., METU

Prof. Dr. Levent Kandiller _____
Industrial Engineering Dept., Çankaya University

Date: 05.05.2010

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 : FATİH RAHİM

Signature :

ABSTRACT

A VARIABLE NEIGHBORHOOD SEARCH PROCEDURE FOR THE COMBINED LOCATION WITH PARTIAL COVERAGE AND SELECTIVE TRAVELING SALESMAN PROBLEM

Rahim, Fatih

M.S., Department of Industrial Engineering
Supervisor: Assoc. Prof. Dr. Canan Sepil

May 2010, 122 pages

In this study, a metaheuristic procedure, particularly a variable neighborhood search procedure, is proposed to solve the combined location and selective traveling salesman problem in glass recycling. The collection of used glass is done by a collecting vehicle that visits a number of predefined collection centers, like restaurants and hospitals that are going to be referred to as compulsory points. Meanwhile, it is desired to locate a predetermined number of bottle banks to residential areas. The aim is to determine the location of these bottle banks and the route of the collecting vehicle so that all compulsory points and all bottle banks are visited and the maximum profit is obtained. Population zones are defined in residential areas and it is assumed that the people in a particular population zone will recycle their used glass to the closest bottle bank that fully or partially covers their zone. A Variable Neighborhood Search algorithm and its variant have been utilized for the solution of the problem. Computational experiments have been made on small and medium scale test problems, randomly generated and adapted from the literature.

Keywords: Neighborhood search, location-routing, traveling salesman problem, maximal covering problem

ÖZ

BİRLEŞİK KISMİ KAPSAMALI YERLEŞİM VE SEÇMELİ GEZGİN SATICI PROBLEMİ İÇİN BİR DEĞİŞKEN KOMŞULUK ARAMA YÖNTEMİ

Rahim, Fatih

Y. Lisans, Endüstri Mühendisliği
Tez Yöneticisi: Doç. Dr. Canan Sepil

Mayıs 2010, 122 sayfa

Bu çalışmada, cam geri dönüşümündeki birleşik yerleşim ve seçmeli gezgin satıcı problemini çözmek için bir metasezgizel yöntem, değişken komşuluk arama yöntemi önerildi. Kullanılmış camların toplanması, restoran, hastane gibi önceden tanımlanmış, zorunlu nokta olarak değinilecek toplama merkezlerinden geçen bir toplama aracı tarafından yapılır. Bu sırada, yerleşim bölgelerine, önceden belirlenmiş sayıdaki geri dönüşüm kumbaralarının yerleştirilmesi istenmektedir. Amaç, bu geri dönüşüm kumbaralarının yerini ve toplama aracının rotasını belirleyerek ve bütün zorunlu nokta ve geri dönüşüm kumbaralarına uğrayarak en yüksek kazancın elde edilmesidir. Yerleşim yerlerindeki nüfus bölgeleri, belli nüfus bölgesindeki insanların kullanılmış camlarını kendi bölgelerini kısmi ya da parçalı olarak kapsayan en yakın geri dönüşüm kumbarasına bırakacakları şekilde tanımlanmıştır. Problemin çözümünde bir değişken komşuluk arama algoritması ile farklı bir varyasyonu kullanılmıştır. Sayısal deneyler, küçük ve orta büyüklükteki, raslantısal oluşturulmuş ya da kaynaklardan uyarlanmış problemler üzerinde yapılmıştır.

Anahtar Kelimeler: Komşuluk tarama, yerleşim-rotalama, gezgin satıcı problemi, maksimum kapsama problemi

To My Family

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Canan Sepil for her guidance, advice, criticism and insight throughout this study. I am very grateful to her for introducing me the subject and for her positive attitude all the time.

I want to thank Assoc. Prof. Dr. Haldun Süral, Assoc. Prof. Dr. Yasemin Serin, Assist. Prof. Dr. Sedef Meral and Prof. Dr. Levent Kandiller for their comments and suggestions that have provided valuable enhancements for my thesis. I want to thank Assist. Prof. Dr. Sedef Meral again for her encouraging me to pursue my master's studies.

I am very grateful to my sister Fethiye Rahim and my mother Yüksel Rahim who are always supporting and motivating me. It would not be possible to carry out this study without their love and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

In this study, our aim is to determine the locations of bottle banks used in collecting recycled glass. In Ankara, currently glass recycling is performed via a limited number of bottle banks provided by ÇEVKO, which is a non-governmental organization established to manage recycling activities. ÇEVKO has worked with the Ministry of Environment and Forestry and located the bottle banks considering the properties of the areas in terms of population and closeness to the industry. Recently, ÇEVKO handed over the possession of the bottle banks to a collecting company in Ankara. The glass collected with the given number of bottle banks, cannot provide satisfactory revenues for the collecting company to compensate for their expenses. The company wants to determine new locations for the bottle banks, with the aim of collecting the maximum amount of glass considering the changes in the population of the residential areas. The company also has some customers, mostly hospitals, restaurants, bars and schools with whom it has made contracts to collect glass on a regular basis. We will refer to these customers as compulsory points that have to be visited every day.

We assume that the company has a fixed number, p, of bottle banks. A limited number of bottle banks are provided by ÇEVKO to the company for free, and the company does not have a control on the number to be located. Currently, because of its financial situation, it does not have the financial means to purchase additional bottle banks and it has to find the best place for these limited resources. There are a number of potential sites to locate these p banks. There are two main factors that affect the desirability of locating a bottle bank in a potential site: closeness of the site to the route of the vehicle that serves the customers and the amount of glass that can be collected in that site. It is required that the vehicle will be routed daily to serve all of the customers and the sites where the bottle banks

are located. The routing part of the problem is the selective traveling salesman problem (STSP) as defined by Gendreau et al. (1998). STSP is defined as the traveling salesman problem (TSP) where the aim is to find a tour of maximal profit that includes all compulsory points. The revenue that will be obtained from a potential site is related with the proximity of the bottle banks to the residences of the droppers. The people will recycle bottles to a bottle bank if it is convenient for them. Therefore, the location part of our problem can be seen as the maximal coverage location problem (MCLP) in the sense that a potential site will cover a population zone, if the proximity to the nearest bank is convenient. Moreover we can include partial coverage as well, and consider the problem as a maximal coverage location problem in the presence of partial coverage (MCLPP), see Karasakal and Karasakal (2004). Considering the two aspects, the problem of determination of the location of bottle banks is done by a special type of the location-routing problem which is a combined selective traveling salesman problem (STSP) and MCLP in the presence of partial coverage MCLPP, i.e. combined MCLPP-STSP.

In Chapter 2, mathematical formulation of the problem and its relation to the MCLPP and STSP are given. In addition, the related literature on location problems and traveling salesman problem is summarized. In Chapter 3, variable neighborhood search (VNS) algorithm is discussed in detail with its literature review. Proposed VNS algorithm and a greedy heuristic are explained in Chapter 4. Computational experiments including the generated problems and experimental results are reported in Chapter 5. Chapter 6 is the conclusion chapter.

CHAPTER 2

MATHEMATICAL MODELLING AND LITERATURE REVIEW

In this chapter, we provide a mathematical model for the combined Maximal Coverage Location Problem with Partial Coverage and Selective Traveling Salesman Problem, MCLPP-STSP. Moreover, the related literature on location problems and TSP problems are reviewed in this chapter.

2.1 Mathematical Model for MCLPP-STSP

Polat (2008) has formulated the MCLPP-STSP for the case when a fleet of vehicles are available to collect recycled glass from the compulsory points and the bottle banks. Here we provide a simpler formulation of the problem for a single vehicle.

Assumptions:

- There are *p* bottle banks to locate.
- There is no fixed cost of locating a bottle bank.
- The bottle banks are assumed to be uncapacitated.
- There are *n* population zones.
- In a population zone, people are assumed to be homogenous in terms of their willingness to recycle glass.
- There are *s* alternative sites to locate the bottle banks and the alternative sites may or may not coincide with the population zones.
- An alternative site (fully or partially) covers a population zone if the zone is within critical distances from the site, or it does not cover the zone outside the critical distances.
- There is a single uncapacitated vehicle.
- There are *m* compulsory points which the vehicle is obliged to visit.

Let N be the set of population zones, K be the set of alternative sites and L_i be the set of alternative sites that can either fully or partially cover zone i. Moreover let M be the set of compulsory points. Thus we can define $H = K \cup M$ as the set of all points that are to be, and can be, visited in a tour. Note that $n = |N|$, $s = |K|$ and $m = |M|$. Since p sites will be selected among s alternative sites to locate the bottle banks, the number of points in a tour of the vehicle will be $f = p + m$.

Parameters:

d_{ij}: distance between points i and j.

c: cost for unit distance traveled.

q: daily expected amount (kg) of recyclable glass that a person can recycle.

r_i: number of residents in the population zone i.

R: unit revenue of glass recycled.

k_{ij}: coverage level (%) defining the rate of recycling in population zone i using alternative site j, $0 \le k_{ij} \le 1$.

w_{ij}: coverage coefficient denoting the total revenue obtained when alternative site j provides coverage to population zone i.

$$w_{ij} = k_{ij} \times r_i \times q \times R$$

p: number of bottle banks to be located.

f: number of points visited by the vehicle.

The coverage level k_{ij} can be determined by using a coverage function as in Karasakal and Karasakal (2004). Hence a population zone i is fully covered by alternative site j if the distance between them is less than distance S, and is partially covered by alternative site j if the distance between them is greater than distance S but less than distance T. In other words, an alternative site can fully

cover zones within the circular field with radius S where it is centered and partially cover zones between the circular fields with radius S and T ($S<T$).

Coverage level provided by alternative site j to a population zone i, defined as the parameter k_{ij}, for $\forall i \in N$, $\forall j \in K$, is given by

$$k_{ij} = \begin{cases} 1 & \text{if } d_{ij} \le S, \\ f(d_{ij}) & \text{if } S < d_{ij} \le T, \\ 0 & \text{otherwise.} \end{cases}$$

Variables:

$$x_{ij} = \begin{cases} 1 \text{ if the vehicle visits point } j \text{ after point } i \\ 0 \text{ otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 \text{ if a bottle bank is located at alternative site } j \\ 0 \text{ otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 \text{ if population zone } i \text{ is fully or partially covered by alternative site } j \\ 0 \text{ otherwise} \end{cases}$$

u_i = label of point i

$$\text{Min z} = \sum_{i \in H} \sum_{j \in H} c\ d_{ij} x_{ij} - \sum_{i \in N} \sum_{j \in Li} w_{ij} z_{ij}$$

Subject to

$$\sum_{i \in H} x_{ij} = 1 \qquad \forall j \in M \qquad (1)$$

$$\sum_{i \in H} x_{ij} = y_j \qquad \forall j \in K \qquad (2)$$

$$\sum_{i \in H} x_{ij} - \sum_{k \in H} x_{jk} = 0 \qquad \forall j \in H \qquad (3)$$

$$u_i - u_j + f x_{ij} \leq f - 1 \qquad \forall i \in H, \forall j \in H \quad i \neq j \qquad (4)$$

$$z_{ij} \leq y_j \qquad \forall j \in L_i, \ \forall i \in N \qquad (5)$$

$$\sum_{j \in Li} z_{ij} \leq 1 \qquad \forall i \in N \qquad (6)$$

$$\sum_{j \in K} y_j = p \qquad (7)$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in H, \forall j \in H \ i \neq j \qquad (8)$$

$$y_j \in \{0,1\} \qquad \forall j \in K \qquad (9)$$

$$z_{ij} \in \{0,1\} \qquad \forall j \in L_i, \forall i \in N \qquad (10)$$

$$u_i \geq 0 \qquad \forall i \in H \qquad (11)$$

The objective function maximizes profit which is composed of revenue gained from the population zones through located bottle banks minus the transportation cost. Since it is a fixed amount, the revenue from the compulsory collection centers is not included in the objective function. Constraints (1) guarantee that the vehicle passes through each compulsory collection point exactly once. Constraints (2) ensure that the vehicle passes through each alternative collection point where a bottle bank is located exactly once and it skips the rest. Constraints (3) ensure that the vehicle leaves the nodes that it visits. Constraints (4) are the Miller-Tucker-Zemlin sub-tour elimination constraints; see Miller et al. (1960). Constraints (5) ensure that if the population zone *i* is fully or partially covered by alternative site *j*, a bottle bank is located there. Constraints (6) ensure that a population zone is covered by at most one alternative point. Constraint (7) ensures that exactly *p* bottle banks are located. While constraints (8), (9), (10) are the integrality constraints, constraints (11) are the non-negativity constraints.

Observe that our model is a combination of two other well known problems: selective traveling salesman problem (STSP) and maximal covering location problem with partial coverage (MCLP-P). Using our notation, Karasakal and Karasakal's (2004) formulation of MCLP-P is as follows:

Maximize z = $\sum_{i \in N} \sum_{j \in L_i} w_{ij} z_{ij}$

Subject to

$z_{ij} \leq y_j$ $\quad \forall j \in L_i, \forall i \in N$

$\sum_{j \in L_i} z_{ij} \leq 1$ $\quad \forall i \in N$

$\sum_{j \in K} y_j = p$

$y_j \in \{0,1\}$ $\quad \forall j \in K$

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

The remaining of our mathematical model is as follows:

Minimize z = $\sum_{i \in H} \sum_{j \in H} c\ d_{ij} x_{ij}$

Subject to

$\sum_{i \in H} x_{ij} = 1$ $\quad \forall j \in M$

$\sum_{i \in H} x_{ij} = y_j$ $\quad \forall j \in K$

$\sum_{i \in H} x_{ij} - \sum_{k \in H} x_{jk} = 0$ $\quad \forall j \in H$

$u_i - u_j + f x_{ij} \leq f - 1$ $\quad \forall i \in H, \forall j \in H\ i \neq j$

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

$y_j \in \{0,1\}$ $\quad \forall j \in K$

$u_i \geq 0 \qquad \forall i \in H$

Note that our constraints (5), (6), (7) belong to MCLP-P and constraints (1), (2), (3), and (4) belong to STSP. Variables of type y_j are common to both of the formulations and provide the link between them. In the case where the daily expected amount of recyclable glass that a person can recycle is zero, since there is no possible profit gained from the alternative sites, our problem reduces to the TSP composed of only the compulsory collection centers. Then, our problem is NP-hard.

Polat (2008), while modeling the multi vehicle version of the problem, has made computational experiments on single vehicle version, the same as ours. She has developed three greedy heuristic procedures. She has generated problems randomly where the maximum number of population zones is 45 for the normal instances and 200 for the large instances. In addition to that she has held a case study for Yenimahalle district in Ankara, Turkey and proposed locations to place bottle banks. Even if her proposed algorithms run very fast, the average percent deviation from optimal for her best algorithm ranges between 1.6 and 5.9 percent and for the worst it is as high as 24.4 percent. However, the performance could be improved by utilizing metaheuristics.

2.2 MCLP Related Literature Review

In the maximal covering location problem, proposed by Church and ReVelle (1974), there are predetermined number of facilities that are going to serve the population within a given distance or time limit from the closest facility. The objective is to maximize the population covered by finding the optimum location for the facilities. The difference from set covering location problem is that, in the set covering problem the aim is to find the minimum number of facilities to locate while all the population is covered. While in MCLP, the goal is to cover the population as much as possible with a given set of facilities. The mathematical formulation of the MCLP is as follows:

Maximize z= $\sum_{i \in I} a_i y_i$

Subject to

$$\sum_{j \in N_i} x_j \geq y_i \qquad \forall i \in I$$

$$\sum_{j \in J} x_j = p$$

$$x_j \in \{0,1\} \qquad \forall j \in J$$

$$y_i \in \{0,1\} \qquad \forall i \in I$$

where I is the set of demand nodes, J is the set of facility sites, a_i is the population at node i and p is the number of facilities to locate. N_i is the set of sites that can serve demand point i, $N_i = \{ j \in J \mid d_{ij} \leq S \}$ where S is the maximum distance for a facility to cover a node and d_{ij} is the distance between node i and j. The decision variables are y_i that represents the coverage of demand point i and x_j represents the location of a facility at node j. The first type of constraints forces a facility located at a site which can serve node i if it is covered. The second type of constraints ensures exactly p facilities are located. The rest are integrality constraints. And the objective is the maximization of the population covered. Another version of MCLP was introduced by Church and ReVelle (1974) that includes the mandatory closeness constraints that forces all the demand points are within a distance, T of the closest facility ($T>S$).

Church and ReVelle (1974) proposed two heuristic algorithms for the solution of MCLP which are called Greedy Adding (GA) and Greedy Adding with Substitution (GAS). In GA, at each iteration a site that covers the highest population that is not covered yet by sites selected in the previous iterations is picked. The algorithm terminates when p facilities are located. The difference of GAS from GA is that at each iteration a substitution move is made by removing a site from the current solution and selecting a new one if it increases the objective.

They have also solved the problem with linear programming, and utilized branch and bound method where necessary.

2.2.1 MCLP with coverage over critical distance

In MCLP, it is assumed that a demand node is covered fully if it is within a distance, *S* of the closest facility or it is not covered at all it the distance is longer. Karasakal and Karasakal (2004) have realized the need for partial coverage for the problems where only full coverage is not sufficient for modeling. They have introduced the notion of partial coverage where a demand point is partially covered by a site if the distance between them is within a minimum (*S*) and maximum critical distance (*T*). If the distance is larger than maximum critical, the demand is not covered at all. In the other case, where the distance is less than the minimum critical distance the demand is fully covered. Note that this problem will reduce to a basic MCLP if *S*=*T*. Samples of possible coverage functions from Karasakal and Karasakal (2004) are given in Figure 1.

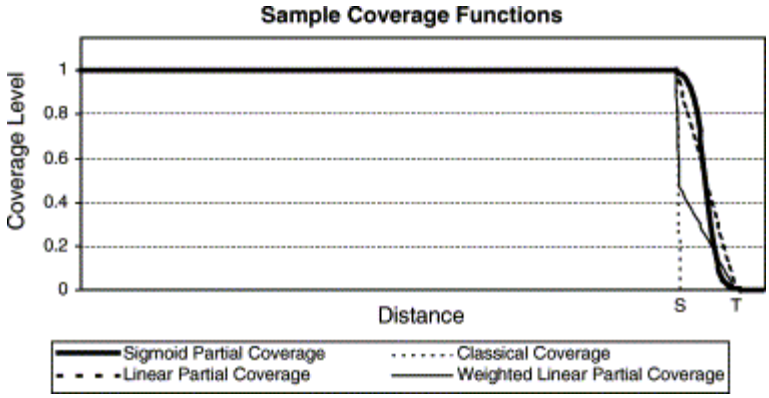

Figure 1. Samples of possible coverage functions.

Then, coverage of a demand point by a site takes the values between 0 and 1 and depends on the distance between them and coverage is higher by the sites closer to the demand point. Karasakal and Karasakal (2004) haven given a new formulation to the MCLP that includes partial coverage and it was depicted in Section 2.1. They have checked the effect of partial coverage by solving the generated

problems optimally for MCLP with and without partial coverage. They have also developed a Lagrangian Relaxation based solution algorithm for large problems and tested it on randomly generated problems.

For better modeling of emergency service facility location problem, Pirkul and Schilling (1989) have made some modifications to the basic MCLP. They have put workload capacities for facilities to prevent excessive workload and introduced the back up service where a demand point should be covered by two facilities, one of which is the back up facility. This way, when the first facility is busy, the demand point will be served by the back up facility. They have also used a service function which is linearly decreasing beyond S with increasing distance between demand point and facility, so they have also allowed partial coverage over critical distance. Level of service for the uncovered demand points is depending on the maximum distance between the facility sites and demand points. In order to solve the problem, they have also used a method based on Lagrangian relaxation. They have made a similar study in Pirkul and Schilling (1991) where only workload capacities are considered. Another study where a partial coverage is allowed is by Church and Roberts (1983) and they have used a piecewise linear step function.

Chung (1986) has given the formulation of capacitated MCLP and shown the formulations of different problems as an MCLP model such as data abstraction, cluster analysis, discriminant analysis, etc. so that the algorithms of MCLP can be used to solve them.

Schilling et al. (1993) have made an extensive survey on covering problems in facility location. They have formed two broad categories which include mandatory coverage models like set covering problem and coverage maximizing models like maximal covering location problem.

Berman and Krass (2002) have introduced a generalized MCLP model where partial coverage is allowed. They have used a decreasing step function of the distance to the closest facility to specify the coverage level. A set of coverage levels and coverage radii have been defined for each demand point. They have developed a greedy heuristic and integer programming formulations and shown its equivalence to the uncapacitated facility location problem without fixed costs.
Batta and Mannur (1990) have formulated a MCLP that includes an importance weight for each demand node and multiple response units required for coverage in order for better modeling of emergency situations.

2.2.2 Solution Algorithms for MCLP

A greedy randomized adaptive search procedure has been proposed by Resende (1998) for the solution of MCLP. He has applied the procedure to a real world problem that is a large telecommunications facility location problem. Galvao et al. (2000) have compared two heuristics based on the relaxation of MCLP, Lagrangean and surrogate relaxations where surrogate relaxation reduces to a 0-1 knapsack problem and the other has the integrality property. Lorena and Pereira (2002) has made a transformation between MCLP and p-median problem utilizing Unified Linear Model and applied a Lagrangean/surrogate heuristic method to solve the MCLP which was originally developed for p-median problem.

Arakaki and Lorena (2001) have applied the constructive genetic algorithm to the MCLP and have made computational experiments on real world and randomly generated problems. Pereira et al. (2007) has formulated MCLP as p-median problem and used a column generation method in order to solve the problem.

2.3 Traveling Salesman Problem with Profits Related Literature Review

Traveling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems. It is to find the shortest route of a salesman who, starting from a home location visits a given set of cities and returns to the

original city given that each city should be visited exactly once. No profit or cost is associated with the cities and the objective is to minimize distance traveled by the salesman. In some variants of the classical TSP, the constraint of visiting all cities is removed and a profit is associated with each city. Then, the salesman can choose the cities that it is going to visit depending on their profit values. Feillet et al. (2005) have named such kind of problems as traveling salesman problems with profits (TSPs with profits). There are two opposing objectives that shape the route of the salesman, to collect profit and to minimize travel costs. While one is forcing to visit more cities the other is pushing to travel less number of cities. They have classified TSPs with profits to three different groups depending on how the two objectives are handled. In the first group, the two objectives are combined in the objective function which is the difference of travel costs and profit collected. DellAmico et al. (1995) has defined this group as the profitable tour problem (PTP). The second group of TSP with profits is called the orienteering problem (OP) where the collected profit is maximized while there is a maximum limit on the total travel cost. In general, the goal in OP is finding a path between two predefined nodes instead of a circuit. The same problem is also named as selective TSP or maximum collection problem. In the third group, there is a minimum limit on the collected profit and the goal is minimizing the travel costs. This problem is called the prize-collecting TSP.

In their formulation of TSP with profits, Feillet et al. (2005) have defined $G = (V,A)$ as a graph where $V=\{v_1,\ldots, v_n\}$ is a set of n vertices and A is a set of edges. A profit p_i is associated with each vertex $v_i \in V$ and a distance c_{ij} is associated with each edge $(v_i, v_j) \in A$. Then, TSP with profits consists of finding a tour where each vertex is visited at most once while both the travel cost and the collected profit are taken into account. v_1 is defined as the depot and it has to be included in the tour. They have formulated the set of constraints common to TSPs with profits as follows:

$$\sum_{v_j \in V \setminus \{v_i\}} x_{ij} = y_i \qquad (v_i \in V), \qquad (1)$$

$$\sum_{v_i \in V \setminus \{v_j\}} x_{ij} = y_j \qquad (v_j \in V), \qquad (2)$$

Subtour elimination constraints, (3)

$y_1 = 1$ (4)

$x_{ij} \in \{0,1\}$ $((v_i,v_j) \in A)$, (5)

$y_i \in \{0,1\}$ $(v_i \in V)$, (6)

where x_{ij} is the binary variable corresponding to each arc $(v_i, v_j) \in A$ and y_i is the binary variable corresponding to every vertex $v_i \in V$. x_{ij} is equal to 1 if the corresponding arc is used in the solution and y_i is equal to 1 if the corresponding vertex is visited. Otherwise, they take the value 0. The first two constraints (1) and (2) are the assignment constraints and the next are the subtour elimination constraints (3). Constraint (4) assures that the depot is included in the tour.

The same constraints apply to PTP and its objective function is

Minimize $\sum_{(v_i,v_j)\in A} c_{ij}x_{ij} - \sum_{v_i \in V} p_i y_i$.

In the OP, there are additional constraints as follows:

$$\sum_{(v_i,v_j \in A)} c_{ij}x_{ij} \le c_{\max},$$

where the objective function is

Maximize $\sum_{v_i \in V} p_i y_i$.

In the PCTSP, the additional constraints are

$$\sum_{v_i \in V} p_i y_i \ge p_{\min},$$

where the objective function is

Minimize $\sum_{(v_i,v_j \in A)} c_{ij}x_{ij}$.

Now we will review the studies made for the solution of OP in the literature.

2.3.1 Exact Solution Procedures for OP

Millar and Kiragu (1997) have proposed an alternative integer-linear formulation of selective TSP which combines permutation variables with flow variables in order to reduce the number of constraints and facilitate the solution of the problems. The proposed method was shown by means of a fisheries patrol problem.

Gendreau et al. (1998) considered the selective TSP where there exist a set of compulsory vertices, T ($T \subset V$, $v_1 \in T$) that has to be included in the tour. Normally, there is only the depot node $\{v_1\}$ which is compulsory. They have proposed two heuristics in order to construct initial feasible solution. In addition to that, they have developed several classes of valid inequalities and used them in a branch-and-cut algorithm.

A similar study was made for OP by Fischetti et al. (1998). They have also proposed a branch-and-cut algorithm based on several families of valid inequalities. They have developed heuristic algorithms to find near-optimal solutions and improve the efficiency of branch-and-cut algorithm.

Righini, Matteo Salani (2006) have addressed a different version of OP which is the OP with time windows (OPTW). In OPTW, a time window and a service time is associated with each vertex. Arrival time to a vertex should be within the time interval and service time is the time spent at the vertex. They have presented a dynamic programming algorithm with decremental state space relaxation for solving the OPTW.

2.3.2 Classical Heuristic Solution Procedures for OP

Tsiligirides (1984) has used the name Score Orienteering Event instead of OP. He has developed a stochastic, a deterministic and a route-improvement algorithms

and compared the performance of the pure stochastic and deterministic heuristics with their combination with improvement heuristic.

Kantor and Rosenwein (1992) have studied the version of OPTW without service time. They have proposed a heuristic method called the 'tree' heuristic which systematically generates a list of feasible paths and then selects the most profitable path.

Chao et al. (1996) has proposed a fast and effective heuristic method based on the notion of record-to-record improvement. It is composed of initialization and improvement steps. Initialization is based on a greedy cheapest insertion method and during the improvement step two point exchange, one point movement and 2-opt procedures are utilized.

2.3.3 Metaheuristic Solution Procedures for OP

Wang et al. (1995) have developed an effective energy function and learning algorithm and based on it they have proposed a modified continuous Hopfield's neural network for OP. The use of traditional heuristics, cheapest insertion and especially 2-opt heuristics considerably improved the performance of the algorithm.

A tabu search heuristic was proposed by Gendreau et al. (1998) for the undirected selective travelling salesman problem. Vertices are inserted to the current tour in clusters and chains of vertices are removed at each iteration. They have used a heuristic which they call Insert and Shake, used to generate the initial solution.

A genetic algorithm has been proposed by Tasgetiren and Smith (2000) for OP. They have compared their algorithm with problem specific heuristics and an artificial neural network. Liang et al. (2002) have applied ant colony optimization which uses local search at each iteration and tabu search algorithm with dynamic

penalty function to search the near-feasible region. Liang and Smith (2006) have also employed an ant colony optimization (ACO) approach for the solution of OP. They have hybridized ACO with VNS, where they have used VNS as a local search procedure at each iteration. It has shown to dominate the heuristics yet proposed.

One should consult to Vansteenwegena et al. (2010) for a recent survey on OP. In the survey, the problem has been classified into four main groups: OP with and without time windows and Team OP with and without time windows. Mathematical formulation, benchmark instances, heuristic and exact solution methods are given for each group. Finally, variants of OP are revised.

2.3.4 Other Studies Related to Selective Traveling Salesman Problem

The main difference of selective TSP from TSP is that it is not required to visit all the cities. There are some other studies that address similar type of problems. Süral and Bookbinder (2003) have first introduced the Single-Vehicle Routing Problem with Unrestricted Backhauls. The problem includes two types of customers: The delivery customers and the collection customers. While the delivery customers receive goods from a depot and have to be visited by a vehicle, the collection customers have to send goods to the depot and it is not required to include them in the tour. The incentive to visit the collection customers is the avoidance of the cost that will be incurred if the goods would be carried to the depot by another means. The aim is to find a route, starting at the depot, visits all delivery customers and selected collection customers and ends at the depot so that the net transportation costs that take into account the cost savings from visiting collection customers are minimized. There is a single vehicle which is capacitated. Süral and Bookbinder (2003) have modeled the problem using the Miller–Tucker–Zemlin (MTZ) subtour elimination constraints. They have studied some techniques for the tight LP relaxations of the model and made it possible to solve medium-sized problems with general-purpose commercial solvers that use branch-

and-bound method. Gutierrez-Jarpa et al. (2009) has addressed the same problem and proposed a branch-and-cut procedure. They have developed two new cuts that handle compulsory and optional customers together, and a new separation algorithm. They have tested their algorithm with the instances of Süral and Bookbinder (2003) and found the results in shorter time. They solved problems with up to 90 customers.

Gribkovskaia et al. (2008) have studied a similar problem, the Single Vehicle Routing Problem with Deliveries and Selective Pickups. In their problem each customer has both delivery and pickup demand instead of only one of them. While the deliveries are mandatory, pickups are optional. A second visit to a customer is allowed so that in case there is not enough space for pickup in the first visit, the vehicle can make a second visit to a customer. The authors have developed a number of construction and improvement heuristics and proposed a Tabu Search algorithm for the problem. The algorithms have been tested on the problems generate from VRPLIB problems.

CHAPTER 3

VARIABLE NEIGHBORHOOD SEARCH

Heuristic search methods are utilized whenever it is not convenient to solve the problems optimally or efficiently with exact algorithms considering the size of the problem. When we have limited time and the problem at hand is NP-hard, it is a good option to use the heuristics that will provide us with near optimal solutions in much shorter time. The most basic of those are the local search algorithms.

Local search algorithms, starting from an initial solution, proceed by making local changes to it each time with an improvement in the objective function value. Eventually, the algorithm terminates when there is no possible enhancement by local changes. The solution found is local optimal but is not necessarily global optimal. In order not to got stuck in a local optimal solution, various metaheuristics have been developed such as simulated annealing, tabu search, GRASP, genetic algorithms, ant colonies, etc. One more recently proposed metaheuristic is the Variable Neighborhood Search (VNS) technique of Mladenović and Hansen (1997). VNS is a simple and effective metaheuristic for combinatorial and global optimization problems that uses systematic change of neighborhood within a possibly randomized local search algorithm.

"Contrary to other metaheuristics based on local search VNS does not follow a trajectory but explores increasingly distant neighborhoods of the current incumbent solution and jumps from this solution to a new one if and only if an improvement has been made. In this way, often favorable characteristics of the incumbent solution, e.g., many variables are already at their optimal value, will be kept and used to obtain promising neighboring solutions. Moreover, a local search routine is applied repeatedly to get from these neighboring solutions to local optima. To construct different neighborhood structures and to perform a

systematic search, one needs to have a way for finding the distance between any two solutions and then induce neighborhoods from it." Hansen and Mladenović (2001).

In this chapter, we give the related literature on the VNS procedure, a detailed description of the procedure, comparison of the method with other metaheuristics and the extensions of the VNS procedure.

Hansen and Mladenović (2001) have provided the details of VNS, which they have developed by combining the idea of changing the neighborhoods in the search with a local search routine. They have listed the rules of the basic VNS, which include developing a neighborhood structure, shaking, local search and move decision. Basic steps of the VNS algorithm can be seen in Figure 2. Different steps of the VNS algorithm are going to be summarized in the following subsections.

Initialization
(1) Select the set of neighborhood structures N_k, for k=1,...,k_{max}
(2) Find an initial solution x;
(3) Select a stopping condition;
Repeat Main step until stopping condition is met
Main Step:
(1) Set $k \leftarrow 1$;

(2) Repeat the following until $k=k_{max}$
(a) Shaking: Generate a solution x' at random from the k^{th} neighborhood of x
($x' \in N_k(x)$);
(b) Local Search: Apply local search method with x' as initial solution;
Denote x'' as the obtained local optimum
(c) Move or not: If this local optimum is better than the incumbent, move there ($x \leftarrow x''$) and go to Step 1;
Otherwise, set $k \leftarrow k+1$.

Figure 2. Basic steps of the VNS algorithm proposed by Hansen and Mladenović (2001).

3.1 Initialization

In the initialization part of the algorithm one has to decide on the neighborhood structure that will be used in the shaking part of the main step. In addition, an initial solution is needed in order to apply the shaking and local search in the main step. The initial solution can either be generated randomly or a good starting solution that will affect the performance of the algorithm positively can be utilized. Finally, a stopping condition should be set so that the search algorithm terminates in a reasonable time. In the following sections, these decisions made in the initialization part are described in more detail.

3.1.1 Neighborhood structures

While most local search heuristics use one, VNS utilize different neighborhood structures. Hansen and Mladenović (2001) has denoted the neighborhood structures as: N_k (k=1,...,k_{max}), a finite set of pre-selected neighborhood structures, and $N_k(x)$, the set of solutions in the k^{th} neighborhood of x. The neighborhood structures are often chosen such that they have higher cardinality as k gets higher. Another way of choosing neighborhoods is that they are successively nested, which means that neighborhood structures with lower index are a subset of the higher ones, $N_1 \subset N_2 \subset \ldots \subset N_{kmax}$. However, according to Blum and Roli (2003), this will decrease efficiency since it may result in revisit of large number of solutions.

In order to define the neighborhood structures we need to measure the distance between two solutions somehow. Let $d(x,x')$ denote the distance function that measures the difference between the solutions x and x'. Then, all x' that satisfy $d(x,x')=k$, i.e. the solutions that are k distant from x, comprise the $N_k(x)$.

Below an example is given for the distance function and neighborhood structures in TSP. The distance between two TSP tours can be calculated by the cardinality of their symmetric difference. *K*-opt neighborhoods by Lin (1965) use this

function. The *k*-opt neighborhood of a solution *x* is composed of all the solutions that differs in *k* links from *x*. In Figure 3, solutions obtained from 2-opt and 3-opt moves of a given tour is shown. Initial solution, *x* is composed of a tour of 6 cities. By removing the links between cities 2, 3 and 1, 6 and reconnecting them, we get the solution x_1. Since *x* and x_1 differs in 2 links, $d(x,x_1)=2$ and all the x_i satisfying this equation compose $N_2(x)$, the 2-opt neighborhood. A 3-opt move is also shown in the figure where three of the links are removed and new ones are constructed.

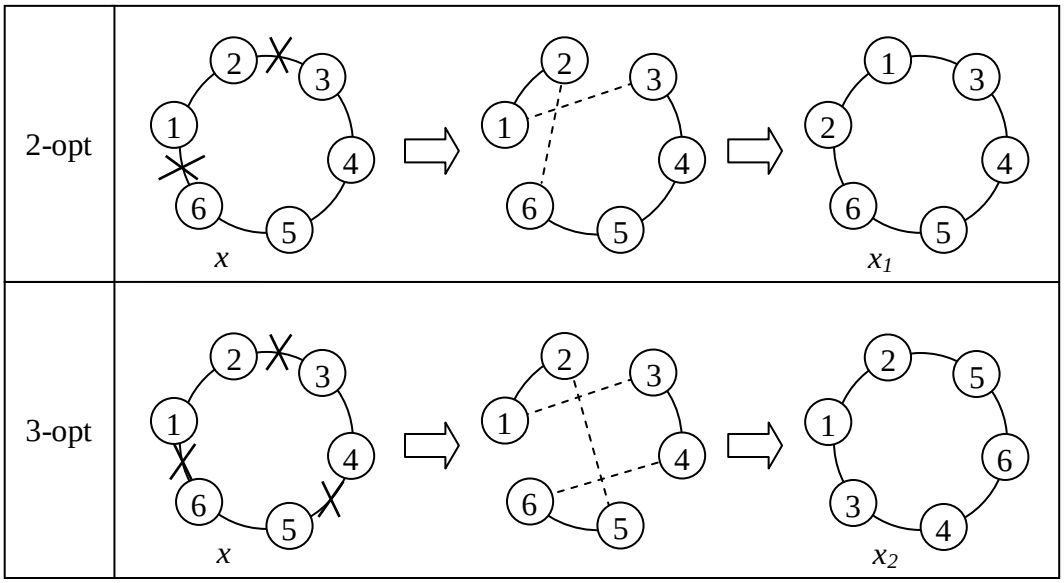

Figure 3. 2-opt and 3-opt moves for TSP.

Observe that maximum number of links to break is 3 if we use non consecutive edges to be broken. And the minimum number of links that will produce a new tour is 2. So the neighborhood structure can be constructed as N_k (k=2,3).

3.1.2 Finding an initial solution

An initial solution of the problem is one of the inputs to the main step of the algorithm. This initial solution can be generated randomly or by a construction

heuristic if it is expected to positively affect the main step so that better solutions can be obtained in shorter time.

3.1.3 Stopping condition

In order to utilize the available time efficiently, a stopping condition should be set with care. The algorithm should run long enough to create near optimal solutions but at the same time it should not continue unnecessarily without making any improvements. In general, the following stopping conditions are used: Maximum CPU time allowed, maximum total number of iterations, maximum number of iterations between two improvements. We can either use one of those conditions solely or we can utilize two or more of them together. For example the algorithm can stop when a preset time limit is reached or if there has been no improvement since a number of iterations determined earlier. Considering the problem structure and availability of time, other stopping conditions can also be developed.

3.2 Main Step

3.2.1 Shaking

Shaking is the stage where neighborhood structure that was defined during initialization is explored. According to the current value of k, the index of the neighborhood, a solution x' is randomly chosen from the $N_k(x)$. If x' is not selected randomly but with a deterministic rule, cycling may occur. The main purpose of shaking is to make perturbations in the incumbent solution so as to escape from local minimum and to provide a good starting point for the local search. The starting point, x', should belong to the basin of attraction of a different local minimum but it should not be too far from x, since otherwise it will be reduced to a random multi-start stated Blum and Roli (2003). Since x' is in the neighborhood of x, some good features x will be kept.

Whenever the local search in the main step does not result in solutions better than the incumbent, the value of k is increased. This helps the algorithm to escape from

the stronger local optimum solutions by making bigger changes in the incumbent solution. So each time local search cannot enhance the incumbent, it has a stronger effect. However, if the local search results in a better solution than the incumbent, then k is set to 1 and search is intensified around the incumbent solution and local search is applied to closest solutions to the incumbent. So regions close to the incumbent solution are explored.

In Figure 4 we can see the effect of shaking and change of neighborhood more clearly. Suppose that we have the objective function graph of a minimization problem and after some runs of the VNS algorithm we have reached the solution x'. There is no chance to escape from that solution by solely local searching since it is a local minimum. In shaking step, perturbations to the current solution is made so that the search is directed to new promising regions while some good features of the current solution is kept. However, at the beginning, since k is small, the perturbations may not suffice to escape from basin of the local minimum, region II, and local search algorithm will find the same local minimum. For example, if x_1 is the result of shaking, then local search will produce the same local optimal x. Any starting solution within the region II will result in x. That is why k is incremented each time local search algorithm cannot find a better solution and larger number of changes are made. Suppose the algorithm has reached x_2 after shaking. Since it is in region I, the local search will produce x'', a local optimal better than x' but still not the global optimal. In the solution space, global optimal is situated far away from x'' and in order to find the optimal solution, shaking should make enough changes to x'' so that local search should start from the basin of the global optimal, region III. This is possible when k is incremented adequately. And if shaking results in a solution like x_3, the algorithm will eventually find global optimal after local search.

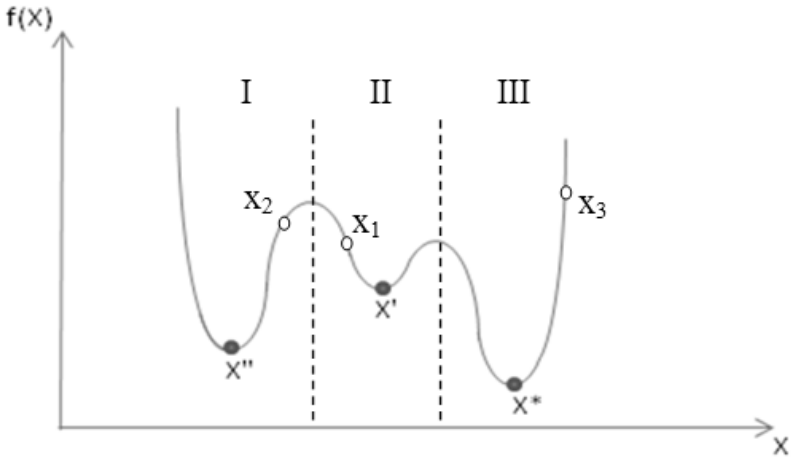

Figure 4. Role of shaking on escaping from local minimum.

3.2.2 Local search

A local search algorithm is applied to the solution generated randomly during the shaking phase. Local search can be based on the neighborhood structure that was defined in the initialization, see Figure 5. The value of k is set to one of the smallest possible values like 1 or 2. Then, the objective function value is calculated for all of the solutions x'' in the neighborhood of x' ($x'' \in N_k(x')$). x'' with the lowest objective function value (for minimization problem) is chosen. If it is lower than the objective function value of x', the solutions in the neighborhood of new solution are evaluated and the procedure is repeated until no better solution can be found (All of the solutions in the neighborhood have higher objective function values than the incumbent). x'' is set to the best solution and it is the result of local search.

Set $x''\leftarrow x'$;

improvement← yes;

Repeat the following until improvement= no

    Calculate objective function value, $f(x''')$, for all $x''' \in N_1(x'')$;

    Set $x'''\leftarrow x'''$ with smallest objective function;

    If $f(x'')>f(x''')$

        Set $x''\leftarrow x'''$;

    If $f(x'')\leq f(x''')$

        improvement← no;

Denote x"as the obtained local optimum

Figure 5. Basic steps of the Local Search algorithm for minimization problem.

This is the simplest local search algorithm that can be used. Considering the problem at hand, more complex and efficient local search algorithms or neighborhood structures different than the one in shaking can be used in order to speed up the overall algorithm.

3.3 Intensification and diversification

Whenever local search cannot make an improvement on the incumbent solution, the neighborhood is changed. This is the source of diversification in VNS, in addition to the shaking itself. Blum and Roli (2003) stated that the choice of neighborhoods of increasing cardinality yields a progressive diversification. As local search does not produce good solutions and k is incremented, in shaking step, x' is selected from neighborhoods of x with higher cardinality and so diversification is increased. In the opposite case, k gets the value 1 and diversification is reduced.

Blum and Roli (2003) explained the effectiveness of changing neighborhoods. They stated that a solution that is locally optimal with respect to a neighborhood is probably not locally optimal with respect to another. The effect of using different

neighborhoods can be seen on Figure 6. Two landscapes defined by two different neighborhoods have been shown. On the landscape on the left local search has stopped at the local minimum $\hat{s}_1$ while it has found a better local minimum $\hat{s}_2$ on the landscape on the right.

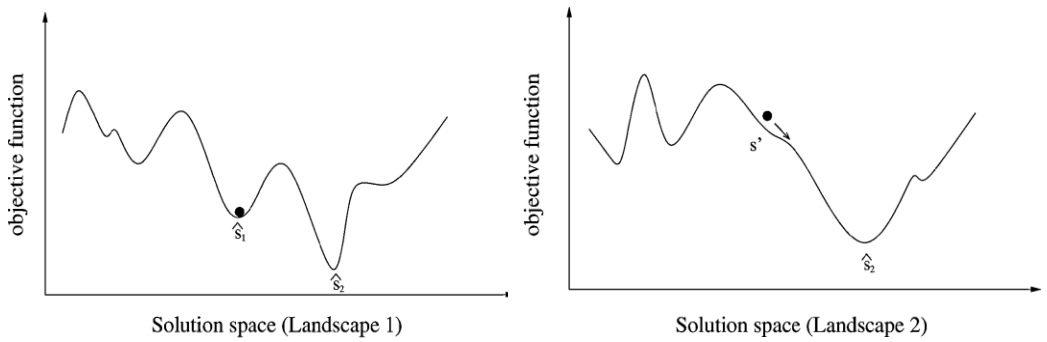

Figure 6. Two search landscapes defined by two different neighborhoods. Blum and Roli (2003).

Hansen and Mladenović (2001) have mentioned achieving intensification and diversification by introducing k_{min} and k_{step} that control change of neighborhoods. In the VNS algorithm, $k \leftarrow 1$ should be replaced with $k \leftarrow k_{min}$ and $k \leftarrow k+1$ with $k \leftarrow k+k_{step}$. Then, diversification is provided by setting k_{min} and/or k_{step} to quite large numbers since the search will continue to regions far from incumbent. In other case, if k_{min} is set to a small integer and $k_{step}=\left\lfloor \frac{k}{l} \right\rfloor+1$, search will stay longer in regions near to incumbent and intensification is achieved.

In order to demonstrate the application and effectiveness of their new algorithm on different problems, Hansen and Mladenović (2001) have solved the traveling salesman problem, the p-median problem, the multi-source Weber problem, the minimum sum-of-squares clustering problem and the bilinear programming problem with bilinear constraints. In order to increase the efficiency and effectiveness of the algorithm in solving large problems, they have proposed three extensions to the basic VNS which are reduced VNS, variable neighborhood decomposition search, and the skewed VNS.

3.4 Parallelization in VNS

In order to increase the speed of the algorithms parallel processing is made where a number of computers are utilized at the same time instead of one. This will provide larger sized problems to be handled and better exploration of the solution space. García-López et al (2002) has studied three different parallel processing for VNS algorithm for the solution of *p*-median problem. The first one is the synchronous parallel VNS where in the local search, neighborhood is partitioned between the processors and the best solution of all the processors is the result of the local search. The aim here is increased speed. In Replicated VNS, where the aim is to explore larger portion of the solution space, each processor solves the problem independently by VNS. And finally in the Replicated Shaking VNS, shaking and local search are performed independently and the best of all results is taken as the new starting point by all the processors. Again, the goal is higher exploration of the solution space. Crainic et al. (2004) applied another parallel processing for VNS which they call cooperative parallel VNS. Here, there is a master process that keeps the overall best solution, starts and ends the algorithm. Each processor executes VNS independently and whenever there is no improvement they continue with the overall best solution received from master process. Again, they have solved the p-median problem. Pérez et al. (2004) summarized the same strategies together.

There are applications of parallel VNS to other problems as well. Polacek et al. (2008) have proposed two new cooperation methods for the parallelization in VNS which includes a self-adapting mechanism for the search parameters. The new methods have been applied to the Multi Depot Vehicle Routing Problem with Time Windows. Sevkli and Aydin (2007) have considered four different strategies for the parallelization of VNS for job shop scheduling problems. They concluded that a noncentral parallelization method with the unilateral-ring topology is the most efficient for the problem considered. Another scheduling problem, Flexible job-shop scheduling with minimizing makespan was addressed by Yazdani et al.

(2010). They have used parallel VNS and different neighborhood structures for a better exploration of the search space.

3.5 Variants of Variable Neighborhood Search

Hansen and Mladenović (2001) proposed some modifications to basic VNS. The first one is to use the notion of simulated annealing. In basic VNS, in step 2c, a move is made only if x'' is better than the incumbent. Like in simulated annealing, we can also accept x'' that is worse than the incumbent with some probability. Then, the new VNS will be a descent-ascent method. Another modification will be moving to the best neighborhood among all. This change will transform the basic VNS to a best improvement method. Three other improvements have been suggested for basic VNS.

Hansen et al. (2001), in order to improve the effectiveness of VNS on solving large problems, have proposed Variable Neighborhood Decomposition Search (VNDS) by integrating basic VNS with successive approximations decomposition method. VNDS is different from basic VNS in the local search phase where a subproblem is solved instead of the whole problem. This new variant of VNS has been applied to the p-median problem and compared mainly with basic VNS, Fast Interchange and Reduced VNS algorithms where it outperformed all in very large instances.

Mladenović et al. (2008) developed a general VNS heuristic for solving continuous global (nonlinear) optimization problems which they call continuous general VNS (CGVNS). They have used different metric functions in the neighborhoods instead of one in addition to the use of different local minimizers. Their main focus is on solving general constrained nonlinear programming problem and to investigate its potential. They have compared CGVNS with GENOCOP III heuristic based on genetic search and other approaches and found that it gives more favorable results.

Puchinger and Raidl (2008) have proposed Relaxation Guided Variable Neighborhood Search which is based on the VNS and VND algorithms. Different than VND, the neighborhoods are changed dynamically instead of having a fixed order. This new variant of VNS has been applied to multidimensional knapsack problem.

Reduced Variable Neighborhood Search (RVNS)

RVNS was proposed in order to get good solutions in short time. The most time consuming part of basic VNS is the local search part. So, local search routine is completely removed and the move step is applied to the solution obtained from shaking step. Each time a solution is randomly picked from the neighborhood and if it is better than the incumbent, search moves to the new solution. Whenever the solution is rejected, k is incremented and the search is continued in larger neighborhoods.

Variable Neighborhood Decomposition Search (VNDS)

Using decomposition with VNS generates VNDS. The difference between basic VNS and VNDS is in step 2b. In the local search step, a subproblem is solved by a local search or other methods instead of solving the whole problem by local search. The subproblem is obtained by fixing all but k attributes of a solution in the shaking step. If the objective function value of the whole solution including the fixed variables and new values of unfixed variables is better than incumbent, we set k=1. Otherwise k is incremented which means the method in 2b is applied to higher number of unfixed variables.

Skewed Variable Neighborhood search (SVNS)

SVNS was proposed to solve the problems where it is necessary to move regions of search space very far from the incumbent. In this kind of cases VNS behaves like multi-start search which is not a very efficient method. In SVNS not only the objective function value of x'' but also its distance from the incumbent is

evaluated when moving to a new solution. A new step is added to the main step of basic VNS that will make use of the distance function. An improvement is made only if $f(x'') < f(x)$ and a move is made only if $f(x'')-\alpha \times d(x,x'')<f(x)$ where $d(x,x'')$ is a distance function and α is a parameter that is used to control the rate of acceptance of a move when $f(x'')$ is greater than $f(x)$. This way the search can also move to solutions that have higher objective value than incumbent but enough distant from it so that the regions far from the incumbent are explored.

Variable Neighborhood Descent (VND)

VND rely on the property that different neighborhoods may have different local optima. Whenever a local optimal solution is found within a neighborhood, local search proceeds from a different neighborhood. In VNS, one neighborhood is used in the local search, but several neighborhoods are used in VND. VND does not include shaking phase. Each time, the best solution in the neighborhood of x is found. If it is better than x, the search continues with the new solution within the same neighborhood. Otherwise, the neighborhood is changed and the local search proceeds with the same solution. See Appendix A for the steps of VNDS, SVNS and VND.

3.6 Use of VNS in Location Problems

Mladenović et al. (2003) addressed the solution of *p*-center problem using Tabu Search and VNS. They have first introduced the vertex substitution method, followed by a Tabu Search where chain-substitution move is utilized. They have developed a VNS algorithm that uses k-interchange move in the shaking step and 1-interchange descent in the local search phase. In the computational runs, it has been shown that in general VNS outperforms Tabu Search and Multi-start Interchange is inferior.

Hansen and Mladenović (1997) have applied the VNS algorithm to another location problem, *p*-median problem that uses fast 1-interchange moves in the

local search phase. They have compared the performance of VNS with Greedy plus interchange and Tabu Search Algorithms and VNS gave better results for average and large problems. They have also made sensitivity analysis for the effect of different values of k_{max}, number of random solutions chosen in the shaking step and k_l and k_{step} that control the change of neighborhoods.

Harm and Hentenryck (2005) has applied VNS to the uncapacitated facility location problem, where a subset of warehouses has to be chosen and assigned to stores so that the fixed costs and the transporation costs are minimized. They have transformed the Tabu Search algorithm with a simple modification to VNS for better diversification. The application of multi-start VNS with 5 replications has produced solutions very close to optimal in short time. Hansen et al. (2007) have addressed the same problem which is also called simple plant location problem. They have used VNDS to solve the primal problem and they have also focused on solving the relaxed dual problem exactly. In order to accomplish that, they have applied VNS to unconstrained dual and then used customized sliding simplex algorithm. In the next phase they have applied branch and bound algorithm that has lead them to find optimal solutions to some large problems.

3.7 Use of VNS in Vehicle Routing Problems

After the development of VNS it has been applied to several different Vehicle Routing Problems recently. Braysy (2003) has used VND in the improvement steps of his four phase solution approach to the vehicle-routing problem with time windows. It was called reactive VNS since it uses the information gathered during the search. Paraskevopoulos et al. (2008) have addressed the Heterogeneous Fleet Vehicle Routing Problem with Time Windows and proposed a Reactive VNS algorithm which is hybridized by Tabu search. Tabu search is used instead of local search which is specially adjusted for intensification. Another VRP with time windows was studied by Polacek et al. (2004). They have applied VNS to

multi depot vehicle routing problem with Time Windows and compared its efficiency with Tabu Search.

Kytöjoki et al. (2007) have focused on solving very large scale vehicle routing problems. They called their proposed algorithm as guided VNS since they have adapted some features of guided local search on VNS. They have used 7 improvement heuristics applied within VND algorithm. Open vehicle routing problem where the vehicles do not return to the depot was addressed by Fleszar et al. (2009) utilizing VNS. Hemmelmayr et al. (2009) have proposed a VNS algorithm for periodic VRP which is the extension of classical VRP for a planning horizon. They have used the same approach for periodic TSP. Finally, a real life case study was made by Wen et al. (2009). The problem includes vehicle routing and driver scheduling in a one week planning horizon. Daily planning problems are solved by VNS after the decomposition of weekly plans.

3.8 Use of VNS in variants of TSP

Three applications of VNS to variants of TSP problem have been found in the literature. Carrabs et al. (2007) have addressed the solution of Pickup and Delivery Traveling Salesman Problem with LIFO Loading using VNS. They have proposed three efficient local search operators with maximum complexity $O(n^3)$ in order to use in VNS. Burke et al. (2001) have proposed a VNS algorithm for Asymmetric Travelling Salesman Problem and used a combination of HyperOpt and 3-opt heuristics in the local search of VNS. In addition to that, they have used which they call “guided shake” in order to prevent the randomness in the shaking phase.

Another application of VNS to TSP variants is by Sevkli and Sevilgen (2006). They have used three different VNS algorithms for the orienteering problem. Two of them are hybridization strategies where RVNS and VND have been used in the local search phase of VNS, and the last is the RVNS.

CHAPTER 4

PROPOSED ALGORITHM

We are going to solve our problem based on the Variable Neighborhood Search (VNS) technique. VNS is a relatively new metaheuristic approach and it has been applied to different combinatorial optimization problems recently. Its application to location problems like, *p*-center and *p*-median have given very good results (see Mladenović et al. (2003) and Hansen and Mladenović (1997)) and we expect to get similar outcome for our problem. We should choose the best alternative locations to place bottle banks and the best route that will pass through both alternative locations chosen and the compulsory collection centers so as to maximize the revenue. When the locations of the bottle banks are given, we can easily find the coverage coefficients or the locations, and determine the revenue from those bottle banks. Then the problem reduces to a TSP. As a consequence, a nested procedure can be applied for the solution of MCLPP-STSP. In the nested procedure, the outer loop should search in the space of alternative sites for locations for the bottle banks, and the inner loop should carry out searches to find the route of the vehicle that will visit all compulsory points and the located bottle banks.

In the outer loop we employ VNS in the space of alternative location sites. While the outer loop of our algorithm implements VNS in order to solve the location problem, in the inner loop, TSP corresponding to each solution to location problem generated by VNS, is solved using Concorde or Linkern, the TSP Solvers by DL Applegate et al. (2006) or using the cheapest insertion heuristic proposed by Rosenkrantz et al. (1977). Now we have to make some definitions before describing our algorithm:

L : Set of alternative collection points where a bottle bank is located.

NL : Set of alternative collection points where there is no bottle bank yet.

x: Solution created in initialization step.

x': Solution obtained in the shaking step.

x'': Solution obtained in the local search.

Noimp: number of iterations without improvement

In order to utilize VNS algorithm we have to find a way to represent a solution. We know that *K* is the set of alternative locations for bottle banks. Then, we divide *K* into two different sets *L* and *NL*. *L* is the set of alternative locations where a bottle bank is located whereas *NL* is composed of the rest of alternative sites in the set *K*. Then, it is clear that the union of *L* and *NL* is equal to *K* (LU *NL=K*). Any subset *L* of the set *K* with the cardinality p represents a feasible solution for the location problem and its element represent the sites to locate bottle banks. The number of such subsets is calculated by the combination of *p* in *s*, $\binom{s}{p}$ where s is the number of alternative collection sites. As the number of bottle banks to locate and the number of alternative locations increase, the number of such subsets, so the number of feasible solutions to the location problem increases very rapidly. For example when *s*=20 and *p*=10, total number of feasible solutions is 184,756. For the problem with *s*=40 and *p*=20, the number of feasible solutions is 1.37847E+11. There is a 746,100 times increase in the size of feasible solution space when the number of alternative sites and bottle banks are doubled.

Now we can explain how we have applied VNS to our problem. While the pseudo code of our algorithm can be seen in Figure 7, the algorithm is described in more detail in the following sections.

Initialization

(1) The set of neighborhood structures N_k k=1,...,k_{max} generated by p function

$p(x_1,x_2)=|x_1 \backslash x_2|=|x_2 \backslash x_1|$, for $x_1,x_2 \in X$

$x' \in N_k(x) \Leftrightarrow p(x',x) = k$

(2) $L := \emptyset$, NL:= composed of all the alternative collection points

For i :=1 to p do

$g \leftarrow$ pick an alternative collection point from NL randomly

$L := L \cup \{g\}$, $NL := NL - \{g\}$

Return x: initial solution

(3) $Noimp$:=0, Calculate $f(x)$ solving the corresponding TSP.

Repeat Main Step until $time > t_{max}$, or $noimp > noimp_{max}$

Main Step

(1) k:=1; $noimp := noimp + 1$

(2) Repeat following until $k = k_{max}$

(a) Shaking

$\{ l_1,...,l_k\} \leftarrow$ pick randomly k elements from L

$\{ nl_1,...,nl_k\} \leftarrow$ pick randomly k elements from NL

Exhange the elements between L and NL

$L := L \cup \{nl_1,...,nl_k\} - \{l_1,...,l_k\}$

$NL := NL \cup \{l_1,...,l_k\} - \{nl_1,...,nl_k\}$

$x' \leftarrow$solution composed of locating bottle banks to sites in L

Solve the TSP corresponding to each x'

(b) Local Search

Repeat the following until $improvement$= no

$improvement$:= no

For all $x'' \in N_1(x')$ do

Solve the TSP corresponding to each x''

Calculate $f(x'')$

If $f(x'') > f(x')$

$x' := x''$, $improvement$:= yes

End if

End for

(c) Move or not

If $f(x') \geq f(x)$ then

k:=1; $x := x'$

Else

$k := k+1$

Figure 7. Pseudo code of VNS applied to MCLPP-STSP.

4.1 Neighborhood structure

In order to define the neighborhood structure, we need to have a distance function first. Suppose a feasible solution, x, is represented by the locations in L. The difference between two solutions x_1 and x_2 is the number of different locations in each solution from the other, which is the Hamming distance. So it is the cardinality of the difference of two sets and represented as follows:

$p(x_1,x_2)=|x_1 \backslash x_2|=|x_2 \backslash x_1|$, for $x_1, x_2 \in X$,

where X is the solution space composed of all possible subsets of K with cardinality p. Then, a neighborhood of a solution x, $N_k(x)$ is composed of all the solutions that differ in k locations from x. A solution is in $N_k(x)$ if it satisfies the following:

$x' \in N_k(x) \Leftrightarrow p(x',x) = k$.

The solutions in the k^{th} neighborhood of a solution, x is reached by replacing k locations in x by k locations in NL set corresponding to x. These moves are called k-substitution move by Mladenovic et al. (1996). Suppose we have a problem with 10 alternative collection points, K={1,2,3,4,5,6,7,8,9,10} and we have 4 bottle banks to locate, p=4. Then, any subset of K with cardinality 4 will represent a feasible solution. A possible feasible solution is, x={1,2,3,4}, and corresponding NL={5,6,7,8,9,10}. $N_1(x)$ is composed of all x' that differs in one location from x. If we substitute "2"and "8" between two sets then we will obtain x'={1,3,4,8} and corresponding NL={2,5,6,7,9,10} which is a 1-substitution move. Observe that $p(x,x')$=1 so $x' \in N_1(x)$. If we substitute 2 locations between L and NL, then we will get a solution from $N_2(x)$, so on so forth. Keep in mind that k_{max} cannot take a value greater than p since substitutions are not possible and we can simply set k_{max} as p.

4.2 The initial solution

We have constructed the initial solutions randomly. At first we have an empty *L* set and *NL* is composed of all alternative collection points. Each time a collection point is chosen and deleted from the *NL* set and added to the set *L* until the cardinality of *L* is *p*. If one wants to start with a better starting point, the greedy construction heuristic that is proposed at the end of the chapter can be used to generate an initial solution.

4.3 Stopping condition

We have employed two stopping conditions. The first one is the iteration limit. The number of iterations of the main step is counted every time k_{max} is reached and it does not produce a better solution than the incumbent. The second stopping condition is the time limit. In order to prevent our algorithm run longer than expected and prevent inefficiency, we have also utilized a time limit. According to the size of the problem and the algorithm, one or both of the strategies have been employed. In the computational experiments section, one can see the stopping condition in more detail.

4.4 Main step

As explained before, the main step is composed of three parts, shaking, local search and move/not move decision. In shaking, the neighborhoods that have defined by *k*-substitution moves will be used. Each time algorithm executes the shaking step, *k* locations in set *L* are randomly picked and replaced by *k* other randomly picked from the set *NL*. The solution we get as a result of shaking is *x'*. In the local search, we have used the neighborhood N_1. We calculate the objective function values of all the solutions in the neighborhood, N_1 of *x'*. These neighborhood solutions are obtained by substituting one location between *L* and *NL*. The search continues starting with the best solution found in the neighborhood. If the best solution found is worse than the new starting solution, the procedure terminates. The newly found solution is *x''*. Now the move decision

is made. If *x''* is better than *x*, the incumbent solution, *x* is assigned the value of *x''* and k is set to 1. So the search is intensified around the new best solution since there will be small perturbations made in shaking. If *x''* is worse than *x*, *k* is incremented and the search is diversified. In the following section, a greedy construction heuristic is proposed that will be compared with VNS in the computational experiments.

4.5 A greedy heuristic procedure

There are different ways of constructing initial solutions. The simplest of all is choosing the locations to place bottle banks randomly from alternative collection points. However, it is not very likely to get good solutions since the solution space is very large. In order to have a good solution in short time, we have proposed a greedy construction heuristic. The heuristic starts from an empty set of locations to place bottle banks, which is denoted by *L*. In each step, a location is chosen from a set of locations, *NL*, that are not selected for placing bottle banks yet. The algorithm terminates until *p* locations have been selected. A similar algorithm was proposed by Church, ReVelle (1974) for the maximal covering location problem. The flow of the proposed greedy heuristic is given in Figure 8. In the initialization step, *L*, *NL*, *Q* and $Profit_L$ are assigned their initial values. Since there are no bottle banks assigned yet, *L* is an empty set and *NL* is composed of all alternative collection points. Since initially the set *L* is empty $Profit_L$ is equal to total profit from compulsory collection centers $Profit_M$. None of the population zones are fully or partially covered yet, so all elements of array *Q* have the value 0. In each run of the main part, all the locations in *NL* are evaluated according to their contribution to the overall revenue. The coverage levels of each location for the population zones that it can serve is compared with the coverage levels already assigned to the population zones and kept in array *Q*. If the location has higher coverage levels than the value in *Q*, the difference is added to the $Profit_{L\cup\{g\}}$ after multiplied by required coefficients. We also have to calculate the cost incurred because of the transportation. Each time we have a new location to add, we have

to solve a new TSP since the new node may change the route previously constructed. So, we solve the TSP that is composed of the locations in L, compulsory collection centers and the alternative point that we are currently evaluating, $L \cup M \cup \{g\}$. The total revenue we get after the inclusion of location $\{g\}$ is denoted by $f(x_{L\cup\{g\}})$ and the location that gives the highest revenue after insertion to set L is denoted by g^*. At the end of each run, L, NL, K and $Profit_L$ are updated. $\{g\}$ is moved from set NL to set L. The coverage levels of zones that have higher rates of coverage by g^* than alternative locations in previous L set are replaced with the new levels. $Profit_L$ is assigned the value $Profit_{L\cup\{g^*\}}$. After all the bottle banks are assigned, the procedure terminates and L is composed of the locations where the bottle banks will be located.

In the first run of the main step, there are s locations to evaluate, so s different TSPs are solved. In the second run s-1 TSPs are solved and in the last run, s-(p-1). Total number of TSPs solved is $s+s\text{-}1+\ldots+s\text{-}(p\text{-}1)$ which is equal to $p\times(s\text{-}(p\text{-}1)/2)$. In the local search, minimum number of TSPs to solve is $p\times(s\text{-}p)$. However, it is the minimum and even if it is smaller than for the greedy heuristic, the main step in local search is repeated several times and in general, it solves more TSPs and run longer than greedy heuristic. The running times and total number of TSPs solved are summarized in Chapter 5.

Let Q, array of size n that keeps coverage levels for each population zone.
Let J_g, set of population zones that are fully or partially covered by alternative collection point g
Let X_L is the solution composed of alternative collection points in L
Let $Profit_L$, profit for set L that is composed of the profit from population zones when they are served by locations in L and the profit from compulsory collection centers.

Initialize L, NL, K and $Profit_L$

(1) $L := \emptyset$, NL:= composed of all alternative collection points.

(2) For i :=1 to $|N|$ do

$Q[i] \leftarrow 0$

(3) $Profit_L \leftarrow Profit_M$

Repeat main step until p number of bottle banks are located:

Main Step

(1) For all $g \in NL$ do

(a) $Profit_{L\cup\{g\}} \leftarrow Profit_L$

(b) For all $n \in J_g$ do

If $k_{ng} > Q[n]$

$Profit_{L\cup\{g\}} \leftarrow Profit_{L\cup\{g\}} + (k_{ng} - K[n]) \times h_n \times q_{person} \times R$

(c) Solve TSP for $L\cup\{g\}\cup M$

(d) Calculate objective function value for $X_{L\cup\{g\}}$

$f(X_{L\cup\{g\}}) = Profit_{L\cup\{g\}} - DistanceTSP_{L\cup\{g\}\cup M} \times c$

(e)If $f(X_{L\cup\{g\}}) > f(X_{L\cup\{g^*\}})$

$g^* \leftarrow g$

(2) Update L, NL, Q and $Profit_L$

(a) $L \leftarrow L \cup g^*$

(b) $NL \leftarrow NL \setminus g^*$

(c) For all $n \in J_{g^*}$ do

If $k_{ng^*} > Q[n]$

$Q[n] \leftarrow k_{ng^*}$

(d) $Profit_L \leftarrow Profit_{L\cup\{g^*\}}$

L is the set of alternative collection points where a bottle bank should be located.

Figure 8. A greedy heuristic procedure to construct initial solution.

CHAPTER 5

COMPUTATIONAL RESULTS

5.1 Problem Sets

In order to test our algorithm, three different sets of problem instances have been utilized. The first and the second sets have been created using the data of TSP and VRP instances reported in TSP Library, TSPlib. The third set of instances has been generated randomly. All three different sets are described in detail in the following sections.

5.1.1 Type 1 Instance Set

These instances are constructed using the problem sets given in TSPlib. Each instance set in TSPlib is specified by the x and y coordinates of the cities and/or the distance matrix. We have chosen burma14, bayg29, dantzig42, eil51, eil101 and KroA200 as our instances. The numbers in these instances indicate the number of cities. We have taken the cities in these instances as the compulsory points of our problem. The additional characteristics of the instances are generated as described in the following paragraphs.

The number of population zones, n, in each instance is determined in accordance with the number of compulsory points, m, in the problem. Thus given m, n is set to three levels of $0.5\times m$, m and $1.5\times m$ for all problem instances other than eil100 and kroA200. For eil100, n is set to 20, 40, 60, 80 and 100 while for kroA200 to 30, 60, 90, 120 and 150. The coordinates of population zones in each instance have been generated independently from a uniform distribution in the ranges defined by each specific problem. The alternative sites coincide with the population zones. The number of bottle banks to locate, p, is determined in

accordance with the size of the problem. Each instance has been solved for different values of p.

S and T values, that define the critical distances for total and partial coverage, affect the solutions considerably. If they are set too high, all alternative collection points will serve almost all population zones and as a result all population zones will be covered 100%. In the other case, total rate of coverage will be too low. In order to prevent both of the extreme cases, S and T values have been set such that in the optimum solution for the problem with highest p value, the ratio of profit gained from alternative collection centers to total possible revenue from population zones is around 90%. This has been achieved by setting S and T, a multiple of the average distance between alternative collection points and population zones. The ratio of S to T has been set as ¾. Using the S and T values, the following function has been used to calculate the coverage level. Coverage level k_{ij} provided by alternative site j to a population zone i, for $\forall i \in N$, $\forall j \in K$, is given by

$$k_{ij} = \begin{cases} 1 & \text{if } d_{ij} \le S, \\ f(d_{ij}) & \text{if } S < d_{ij} \le T, \\ 0 & \text{otherwise,} \end{cases}$$

where $f(d_{ij}) = \dfrac{T - d_{ij}}{T - S}$ for $S < d_{ij} \le T$.

Figure 9 shows the coverage level as a function of the distance.

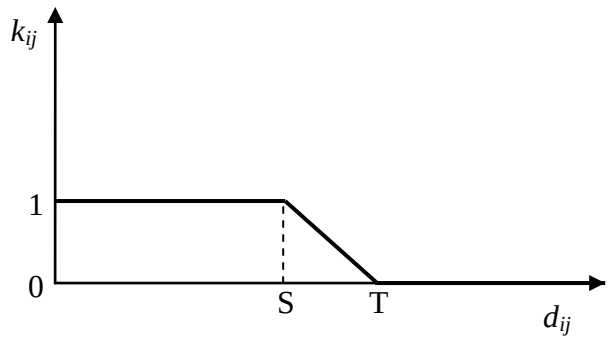

Figure 9. Graph of k_{ij} with linear partial coverage function.

Daily amount of recyclable glass that a person can leave, q, is set to 0.05 kg and revenue, R, gained from 1 kg of recycled glass has been set to 0.1. Since the distances differ greatly from instance to instance, using the same cost coefficient for all problems will cause unstable results. In order to make profits and costs consistent with each other, different cost coefficients have been calculated for different problems. TSPs composed of only the compulsory collection centers have been solved for each problem and cost coefficient is set so that the transportation cost is half of the average total revenue from compulsory collection centers. Thus, following equation has been used in order to calculate cost per unit distance, c, where A is the set of pairs (i,j) that belongs to the optimum TSP tour of compulsory collection centers.

$$c = \frac{q \times m \times R}{2 \times \sum_{(i,j) \in A} d_{ij}}$$

Finally, the amount of glass to be picked up daily from compulsory collection centers, q_j, $j \in M$, have been generated randomly from a discrete uniform distribution between 100 and 200, and the population in each residential zone has been generated randomly from a discrete uniform distribution between 400 and 800.

5.1.2 Type 2 Instance Set

Type 2 instance set is constructed using the VRP instances in TSPlib. Instances named eil22, eil23, eil30, eil33, att48, eil76, eil101 and gil262 are considered. For each of the instances, three different problems have been generated. In the first problem, there is only one compulsory point, which is the depot in the corresponding VRP instance. The second and third kind of problems are generated such that the first 15% and 30% of the nodes of VRP instance are assigned as the compulsory collection points in our problem and the rest are assigned as alternative collection centers that will serve the population zones. Therefore, the coordinates of compulsory collection points and alternative collection points are

taken from the VRP instances. n is set as $3\times s$. The coordinates of population zones are generated from a uniform distribution in the coordinates range of alternative collection centers. Consequently, the population zones are close to alternative collection points regardless of the location of compulsory collection points. The rest of the parameters are generated as in the first type of instance set.

The difference between Type 1 and Type 2 instance sets is that in Type 2 problems not all the points of the VRP instance are included in TSP tour while in Type 1 problems all the cities of TSP instance are visited in addition to the alternative collection points where a bottle bank is located. Moreover, the ratio of the number of alternative collection points to compulsory collection centers is different. While in Type 1 problem sets this ratio differs between 0.5 and 1.5, in Type 2 sets it is between around 2.3 and 5.7.

5.1.3 Type 3 Instance Set

In this set, the coordinates of the population zones have been generated randomly from the uniform between 0 and 1000 and the collection points, both compulsory and alternative, coincide with the population zones. s is selected as 20, 30, 40, 50, 60, 80 and 100 while m is set to $s/2$, and n is set to $3\times s$. The rest of the parameters have been set as in Type 1 instances.

5.2 TSP Algorithms

Each time we have a solution to the outer problem, we have to solve a TSP that includes both bottle bank locations and compulsory collection points. It is vital to employ an efficient algorithm that will solve the TSP near to optimal quickly since it will be called as many times as the VNS algorithm creates solutions to the location problem. Three of the state of the art algorithms that were developed for TSP are Concorde TSP solver, Linkern and LKH.

LKH algorithm is based on an effective implementation of, Lin-Kernighan heuristic developed by Helsgaun (2000). Its main difference from basic Lin-Kernighan heuristic by Lin (1973) is the use of more complex search steps and sensitivity analysis to control the search. The Chained-Lin-Kernighan heuristic which was proposed by Applegate et al. (2003) has been utilized in Linkern algorithm. Concorde is an exact TSP solver based on a cutting-plane method, see Applegate et al. (2000).

In order to test their efficiency in solving our generated problems and decide on which one to use, preliminary runs have been made using problems from TSPlib. CPU times are plotted in Figure 10. As can been seen, Concorde Algorithm has much higher CPU times than the other two, and Linkern algorithm runs slightly faster than LKH. The differences in running times are getting larger with increasing number of cities and for the largest problem, kroA200, CPU time of Concorde is 4.9 times that of Linkern and 2.7 times of LKH. If we compare the lengths of the routes found by Linkern and LKH with the tour length of Concorde, we see that both of them have found optimum tours for all of the problems except ch150 and percent error for this instance is less than 0.02.

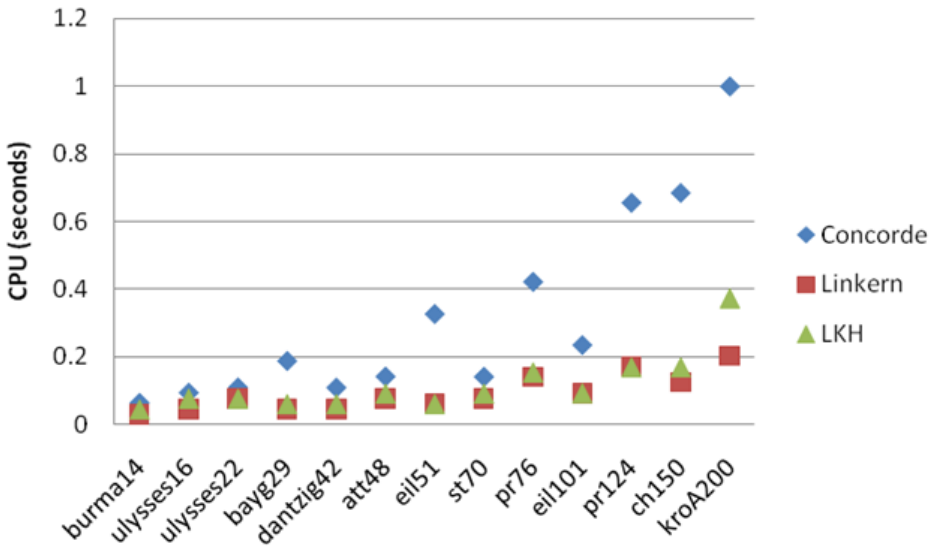

Figure 10. CPU times of TSP Algorithms for 13 TSP problems of TSPlib.

We have made higher number of experiments with VRP instances. First, the TSP Algorithms have been tested with the original VRP problems. 9 of the VRP problems of TSPlib ranging from 22 to 262 cities have been solved and run times have been given in Figure 11. Once again, since it is the only exact algorithm we have tested, Concorde Algorithm has the highest running times. In the largest instance, gil262, it runs 6 times longer than LKH and 2.3 times longer than Linkern. And Linkern runs slightly faster than LKH Algorithm for VRP instances as well. If we check the distances obtained, it is seen that both Linkern and LKH Algorithms have solved the TSPs with VRP data optimally.

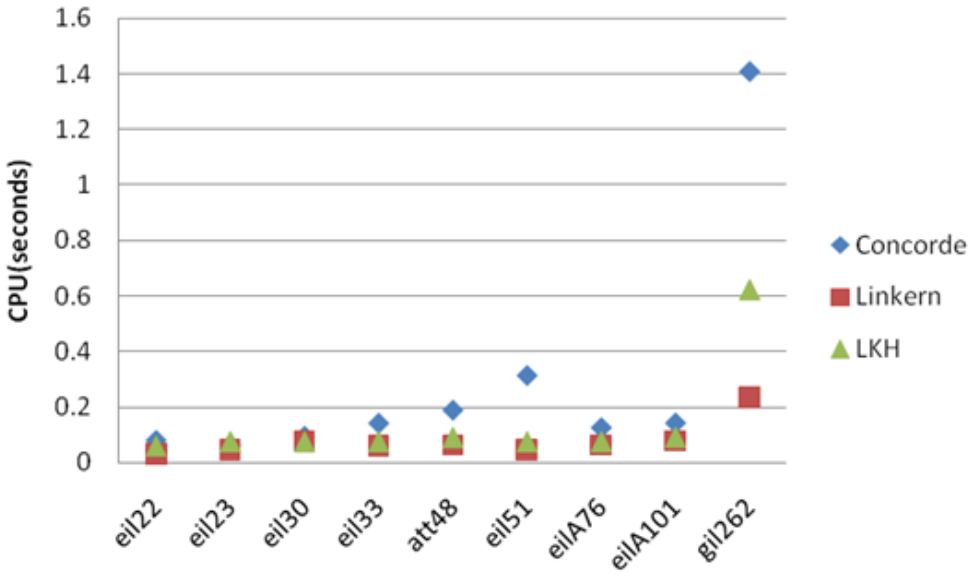

Figure 11. CPU times of TSP Algorithms for 9 VRP problems of TSPlib.

However, in our original problem, we make some modifications to the TSP and VRP instances of TSPlib. We solve TSP problems created by adding some randomly generated cities to the original problem and excluding some of cities in it. In order to check the performance of the TSP Algorithms in our original problem, we have solved 100 TSPs for each problem we have generated from VRP instances. These TSPs have been created considering the highest number of bottle banks to locate for the corresponding problem. For example, for the instance, eil22, m=2, s=20, and p=4, 6, 8, 10, there are 2 compulsory collection

points. Minimum and maximum numbers of bottle banks to locate are 4 and 10. So we have constructed 100 TSPs by choosing each time 10 locations from 20 possible in addition to two compulsory nodes. In Figure 12 Average CPU time of each 100 TSPs with same length is summarized. The results are similar to the ones we got for original TSP and VRP problems, Concorde is the slowest and Linkern is the fastest algorithm once again. LKH runs slower than Concorde for the small TSPs but the problems greater than 30 cities run time of Concorde is higher than the other two. 2,700 different TSP problems have been generated and run by each of the Algorithm and Linkern has found the optimum for all except 7 of them and LKH found all but 14 of them. Maximum average errors in 27 different problems are 0.0084 and 0.0096 for Linkern and LKH, respectively. The CPU times and percent errors of three different algorithms can be found in Appendix B for the solution of TSP and VRP from TSPlib and the generated problems from VRP data.

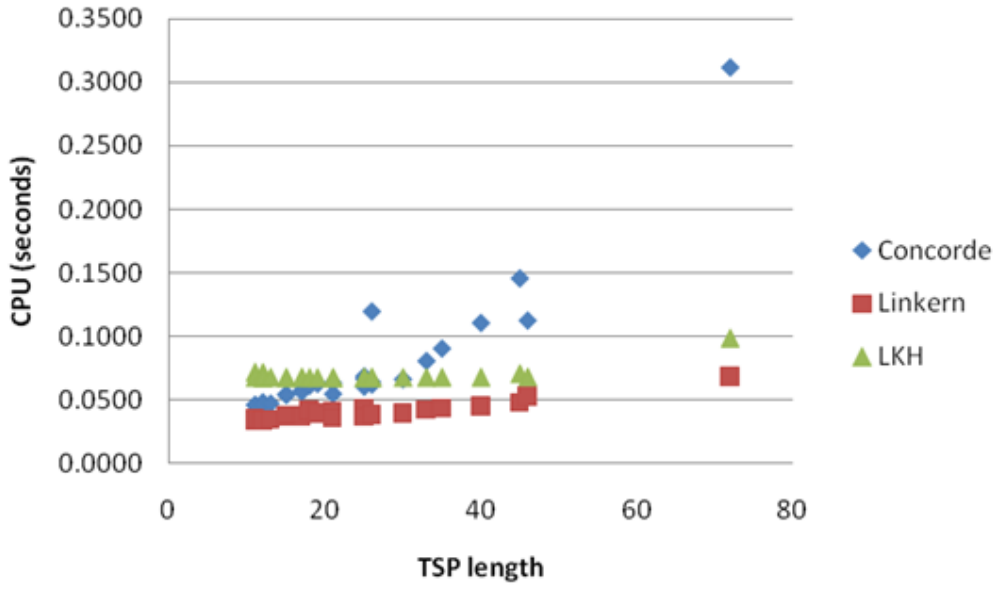

Figure 12. Average CPU times of TSP Algorithms for problems generated from VRP instances.

Considering both running times and quality of solutions found together, Linkern Algorithm outperforms the other two since it finds optimal or near optimal

solutions faster than both of them. From now on, Concorde TSP solver will be used for exhaustive search where optimum solution is found. For greedy heuristic and local search algorithms, Linkern will be utilized with TSPs greater than 8 cities. We have also used Concorde in the heuristics where the size of corresponding TSP is less than 8 since Linkern does not solve very small instances. For VNS, we will first utilize the Concorde and Linkern TSP solvers like in the heuristic algorithms and we will compare their efficiency with the cheapest insertion and 2-opt heuristics.

5.3 Parameter Setting for VNS using Linkern

VNS is a metaheuristic that has very few parameters to set and it is very straightforward to start implementation. We have already decided on neighborhood structure to be used in the shaking step and local search. Now we have to decide on k_{max} and stopping condition. In our problem, k_{max} and stopping condition is critical, since each time an objective function is calculated, we also have to solve a TSP problem and running the algorithm unnecessarily without any improvement will make it inefficient. As a stopping condition we will use an iteration limit. The aim is not stop the algorithm with a time limit whenever it is possible to make more improvements. We could also use maximum number of iterations without improvement. In order to decide on the maximum iteration number and value of k_{max}, preliminary runs have been made.

Preliminary runs have been made for type 3 problems, ran50 and ran100. k_{max} has been given 3 different values, 3, p and $p/2$. In addition to that, one more run has been made with $k_{max}=p$ and using a step greater than 1. The step values used for different p can been seen in Table 1. Since we do not have p values much greater than 20, we did not set higher values for step. Setting step greater than 1 will increase the diversification since it will make larger perturbations to the current solution more frequently and will make the main step execute faster.

Table 1. Value of step for different *p* values.

p	step
p<10	1
10≤*p*<15	2
15≤*p*<20	3
20≤*p*	4

We have made runs long enough for each k_{max} value and *p* combination so that the algorithm converges. Figure 13 shows the evolution of the algorithm for different k_{max} values during the solution of the problem ran100 with 15 bottle banks to locate. Eventually, all of the runs converged to the same number. While k_{max}=3 was the fastest, k_{max}=*p* was the slowest of all to converge to the best solution found.

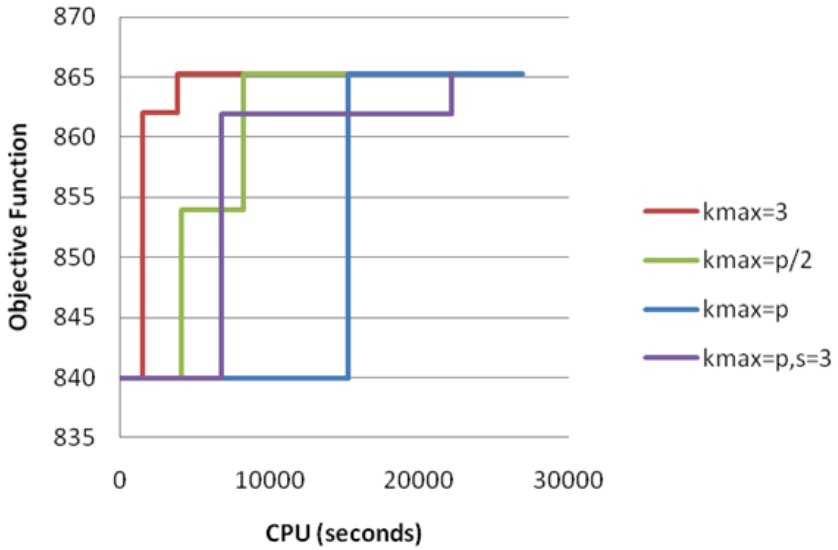

Figure 13. Progress of VNS algorithm with different k_{max} values, for ran100, *p*=15.

After evaluating the results we got the Table 2 that shows the CPU time at which each combination reaches the overall best solution. "*" indicates that the algorithm did not converge to the best solution found in the set time limit.

Highlighted CPU shows the k_{max} value that reaches the best solution in shortest time. So in half of the instances, algorithm was fastest when k_{max} is 3 and in 2 of the instances, it was fastest when k_{max} is p and step is used and for one instance it was fastest when k_{max} is $p/2$.

Table 2. Minimum CPU required to find the best objective for different k_{max} values.

	ran50			ran100		
	10	15	20	10	15	20
k_{max}=3	2075	7178	626	1097	3904	*
k_{max} =p/2	521	2548	2657	1976	8287	30966
k_{max} =p	2171	5142	10853	4839	15291	55605
k_{max} =p,*step*	854	2051	4426	2481	22160	12599

* not converged

The reason for the speed of the algorithm with k_{max}=3, and $p/2$ maybe the strength of the local search itself. However, setting k_{max} these values may not provide enough diversification for larger problems and may cause longer runs or worse results in the same time limit. In addition to that, setting k_{max}=p will substantially increase the running time even if it will eventually give good results. Consequently, we have adopted the last strategy where k_{max}=p and step is greater than 1 for the VNS algorithm that uses Linkern to solve TSPs generated.

However, even for the best parameters we have adapted, the CPU times are very high for a heuristic algorithm that does not guarantee optimality. The CPU time for the problem ran100 with 20 bottle banks is 9.5 hours and around 6 hours for the VRP instance eil101 with 25 bottle banks. The most time consuming part of our algorithm is the solution of the TSP problems generated. There are two main steps where TSP problems are created, shaking step and local search. Each time the shaking step is executed, k alternative sites are removed from the current solution and k new sites are introduced to place bottle banks. Then, k nodes are removed from the current vertex set of TSP and k new nodes are introduced. The

rest of the vertices stay unchanged. In the local search step, there is only one node deleted and a new one added to the vertex set since each time a substitution of sites is made. So instead of using Linkern and Concorde each time there is a change in TSP tour, we could utilize construction and improvement heuristics to shorten the CPU time. We have evaluated three different alternatives as in Table 3 and compared them with the use of Linkern.

Table 3. Alternative methods used for the solution of TSP in VNS.

	VNS-L	**VNS-L-I**	**VNS-I-I**	**VNS-I-2-I**
Shaking	Linkern	Linkern	Cheapest Insertion	Cheapest Insertion + 2 opt
Local Search	Linkern	Cheapest Insertion	Cheapest Insertion	Cheapest Insertion

Cheapest insertion is a construction heuristic and it requires a subtour for the insertion of new nodes to the tour. In the shaking step, subtour is obtained using the TSP tour of the current solution. The sites deleted from the solution of location problem are removed one by one from the TSP tour by connecting the adjacent nodes of the deleted node each time. A total of *k* nodes are excluded from the tour and the subtour is obtained. In the local search, only one node is deleted from the tour. After the subtour is obtained, newly chosen sites to locate bottle banks are inserted between the nodes that will cause the smallest increase in the TSP tour distance as in the cheapest insertion heuristic. While in shaking step *k* new nodes are added to the subtour, in local search only one is added. Because of the larger change in the TSP tour in shaking step, and cheapest insertion may not suffice to produce good results, we have also tried to improve the tour obtained as a result of cheapest insertion by using 2-opt heuristic. 2-opt is an improvement heuristic proposed by Croes (1958) that finds the best tour by breaking two edges at a time and generating a new tour by connecting the resulting paths in opposite way. A move is made if a shorter tour is found and the search continues with the new tour. The algorithm terminates when there is no more possible improvement.

The performance of VNS algorithm that only uses Linkern and Concorde has been compared with the three new alternatives by solving problems from each instance set. In Table 4, percent deviation from best solution and CPU values are given for three randomly generated problems. If we consider the solution quality, VNS-L-I and VNS-L have produced the same results, finding the best solution for all of 12 problems. The other two alternatives have worse solution results, finding only 2 and 3 of the best solutions with maximum error 7.96 and 1.93. If we look at the CPU times, VNS-L, runs much slower than the alternative methods. For the largest problem, ran100 with 20 bottle banks, CPU time is around 9.5 hours for VNS-L, while it is only 109.22 seconds for VNS-L-I and 16.13 seconds for VNS-I-2-I. The reason for the better performance of VNS-L-I than VNS-I-I and VNS-I-2-I is its use of Linkern in shaking step instead of the cheapest insertion and 2-opt heuristics. Since there is a larger change in the TSP tour in shaking step, those two heuristics does not suffice to produce as good TSP tours as Linkern. And the reason for slowness of VNS-L is its use of Linkern also in local search. Even if there is only one node change in the TSP tour, it solves the problem from scratch. However, insertion heuristic is able to produce the same tour in much shorter time.

Table 4. CPU Time and % Error of VNS with alternative TSP solution procedures for the random instances ran20, ran50 and ran100.

				% Deviation from Best				CPU (seconds)			
Problem set	*m-n-s*	*p*	f(best)	VNS-L	VNS-L-I	VNS-I-I	VNS-I-2-I	VNS-L	VNS-L-I	VNS-I-I	VNS-I-2-I
ran20	10 60 20	4*	50.54	0.00	0.00	2.01	0.00	26	0.31	0.09	0.03
		6*	76.23	0.00	0.00	1.71	0.00	46	0.44	0.05	0.05
		8*	88.97	0.00	0.00	7.96	1.22	64	0.95	0.06	0.05
		10*	94.20	0.00	0.00	0.00	1.52	113	0.59	0.06	0.06
ran50	25 150 50	5*	329.28	0.00	0.00	0.97	0.00	129	0.33	0.08	0.08
		10	442.02	0.00	0.00	1.09	0.14	1122	2.14	0.47	0.45
		15	502.63	0.00	0.00	1.30	0.67	4498	3.28	2.39	1.80
		20	529.08	0.00	0.00	0.37	0.93	2225	3.67	2.86	4.36
ran100	50 300 100	5	490.17	0.00	0.00	0.00	0.51	675	0.89	0.27	0.27
		10	708.82	0.00	0.00	1.35	0.56	2693	9.67	6.63	1.77
		15	865.23	0.00	0.00	2.78	1.50	4963	31.27	11.25	32.78
		20	952.92	0.00	0.00	1.72	1.93	34460	109.22	17.70	16.13
			Average	0.00	0.00	1.77	0.75	4251.17	13.56	3.49	4.82
			Maximum	0.00	0.00	7.96	1.93	34460.00	109.22	17.70	32.78
			# Best	12	12	2	3				
			Total	12							

The results are similar for the TSP and VRP instance sets, see Appendix C. VNS-L-I produces as good results as VNS-L in much shorter time and gives better results than the other two alternatives in comparable time. So we have adapted using the first alternative we have proposed, using Linkern in shaking step and cheapest insertion heuristic in local search step. If not stated explicitly, from now on VNS will represent this new combination of heuristics.

5.4 Parameter Setting for VNS using Cheapest Insertion Heuristic

We have used p as k_{max}, 1 as k_{min} and 1 as k_{step} in the previous comparisons for VNS-L-I. In order to increase the performance of our new algorithm we have made a more extensive study for parameter setting for VNS-L-I than for VNS-L. We have tried $p/2$, $3\times p/4$ and p for k_{max} and 1, $p/4$ and $p/2$ for k_{min}. We have also tried similar values for k_{step} like in VNS-L, where k_{step} is a bit higher for smaller k_{max}-k_{min} values, see Table 5.

Table 5. Value of *step* for different k_{max}-k_{min} values.

k_{max}-k_{min}	k_{step}
≤ 5	1
5 < ≤ 10	2
10 < ≤ 20	3
20 <	4

A total of 13 alternatives have been evaluated by solving 18 problems and each alternative has been run with an iteration limit of 100 for the main step. The iteration number of main step at which the best solution found is counted in order to calculate the iteration limit. If we choose three standard deviations from the average iteration limit, the best solution is found for almost all the alternatives. When the CPU times are evaluated, the alternative with $k_{max}=3\times p/4$, $k_{min}=1$ and $k_{step}>1$ has been found out to be the fastest one to give best results. So we have adopted this strategy with an iteration limit for the main step set to 40.

5.5 Computational Results

The proposed algorithm has been coded in C programming language and it has been run on computers with Intel Core 2 Duo 3.00 GHz CPU processor and 3.49 GB of RAM. In this section, first the computational results for the three different problem sets have been discussed. The performance of VNS algorithm has been compared with pure local search algorithm, proposed Greedy Heuristic and where applicable, with the optimal solutions obtained by exhaustive search. Later, the results for the large instances have been given. RVNS algorithm has also been applied to large instances as it was proposed to get good results in short time. In order to compare the performance we have used the following performance measures: CPU Time, percent deviation from optimum or best and percent of all TSPs solved. Percent deviation is calculated as follows:

$\%\ Dev_{opt} = \frac{f_{opt} - f_{cur}}{f_{opt}} \times 100,$

$\%\ Dev_{best} = \frac{f_{best} - f_{cur}}{f_{best}} \times 100,$

where f_{opt} indicates the objective function value of the optimum solution, f_{best} is the objective function value of the best found solution and f_{cur} is the objective function value of the current solution found. Note that in some of the problems since the ratio of the revenue obtained from the compulsory collection points to the maximum possible revenue that can be obtained with a given number of bottle banks is high, the solution results may be overrated if we use the profits that include revenue from both collection points. In order to prevent that we subtract the profit obtained from the solution of the problem with no bottle banks, from overall profit and use it in calculation of % deviation.

Percent of TSP solved is calculated with the following formula:

$\%\ \text{TSP solved} = \frac{\#\ of\ TSP\ solved}{Total\ \#\ of\ TSPs} \times 100$

Total number of TSPs is all possible selection of locations from alternative collection points in order to locate the bottle banks. It is the combination of p in s.

The following notation has been used in the tables of solution results:

GH: Greedy Heuristic Algorithm

LS: Local Search Algorithm

VNS: Variable Neighborhood Search Algorithm

ES: Exhaustive Search

L: Linkern TSP solver

C: Concorde TSP solver

I: Cheapest Insertion Heuristic

Profit(0): Profit gained from only compulsory collection points

Profit(p): Contribution of locating p bottle banks to the overall profit

(Profit(p)= Overall Profit-Profit(0))

Linkern TSP solver has been used for the solution of TSPs generated by LS, GH and shaking step of VNS in order to calculate the objective function value. However, for the TSPs with less than 9 cities, Concorde TSP solver has been utilized since Linkern is not designed to solve small instances. It has been stated in the tables in the appendices which ones have been used. Whenever possible, optimum solutions have been found for evaluation of the proposed algorithms by ES which evaluates all the possible combinations and uses Concorde TSP solver. A time limit of 40 hours has been set for ES. Now we can discuss the results obtained for different set of problems.

5.5.1 Results for TSP instances

In Table 6 computational results for the TSP instances are summarized. In the first column, the name of the problem instance is stated. In the next column the number of compulsory collection points, population zones and alternative sites are given. Each problem has been solved with several different number of bottle banks and these numbers are listed in the column p. Then, comparison of different algorithms with respect to percent deviation from the best found solution, the computation time and percentage of all possible TSPs solved are given. Note that these numbers are the average of the solutions of a problem with different p.

Table 6. Summary of computational results for TSP Instances.

			% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
burma14	14-7-7	3	1.51	0.00	0.00	0.73	1.00	0.80	2	51	69
	14-14-14	4,5,6,7	5.26	3.03	0.00	2.43	10.00	0.30	144	3	13
	14-21-21	4,6,8,10	2.98	0.57	0.00	4.64	17.75	0.57	10227	0	1
bayg29	29-15-15	4,5,6,7	0.00	0.00	0.00	2.99	10.50	0.61	409	2	7
	29-29-29	4,6,8,10,12,14	1.24	1.30	0.00	9.56	55.67	1.23	28605	0	0
	29-45-45	4,8,12,16,20	0.40	0.19	0.00	21.97	210.60	2.02	14781	0	0
dantzig42	42-21-21	4,6,8,10	0.00	0.90	0.00	5.84	29.25	0.77	20962	0	2
	42-42-42	4,8,12,16,20	0.00	0.29	0.00	20.71	141.00	0.49	10786	0	0
	42-63-63	4,8,12,16,21	1.02	0.28	0.00	35.39	351.00	1.39	69081	0	0
eil51	51-25-25	4,6,8,10,12	0.01	1.33	0.00	8.33	49.00	0.86	28909	0	1
	51-51-51	4,8,12,16,20,24	1.17	0.13	0.00	32.91	270.67	1.03	71859	0	0
	51-75-75	4,8,12,16	4.11	1.33	0.00	39.79	410.00	3.24		0	0
eil101	101-20-20	4,6,8,10	0.00	0.00	0.00	10.56	40.25	0.67	32707	0	1
	101-40-40	4,8,12,16,20	0.51	0.49	0.00	35.28	244.20	1.48	39287	0	0
	101-60-60	4,8,12,16,20	0.78	0.39	0.00	58.55	685.40	3.45		0	0
	101-80-80	4,8,12,16	1.31	0.90	0.00	65.71	800.00	5.52		0	0
	101-100-100	4,8,12,17	1.52	0.00	11.35	89.61	925.75	7.32		0	0
		Average	1.28	0.65	0.67	26.18	250.12	1.87	25212	3	6
		Maximum	5.26	3.03	11.35	89.61	925.75	7.32	71859	51	69

VNS algorithm has found the best solutions for all the TSP instances. While the maximum percent deviation is similar for LS and GH, LS algorithm performs better in general than GH, if we look at the average percent deviation from the best solution. When % TSP solved is considered, it is seen that as the problem size decreases, the efficiency also decreases for greedy heuristic and local search. For the smallest problem, burma14 with 7 alternative sites, GH and LS has solved 51% and 69% of all possible TSPs respectively. As the problem size increases this ratio decreases dramatically. VNS is the fastest of the three algorithms while LS is the slowest. In average VNS is 13.5 times faster than GH and GH is 9 times faster than LS. The reason is that GH is an insertion heuristic where exactly *p* evaluations are made to construct the final solution and iteration proceeds in LS while there is progress in the objective value. The reason for VNS being fast is that even if it solves much higher number of TSPs, it uses cheapest insertion heuristic instead of Linkern in the local search.

The objective function values, computation times and percent deviation from optimum and best found solutions for each problem instance are given in Appendix D. There are a total of 71 solutions for TSP instances that includes different number of bottle banks. VNS algorithm has given the best results for all of them while it is 30 for GH and 47 for LS. Both GH and LS also performed well even if they do not always find optimum solutions. The maximum deviation from the best is 6.74 for GH and 12.11 for LS considering all 71 problems.

5.5.2 Results for VRP instances

In Table 7, the results for the VRP instances are summarized. While VNS has found the best solutions for all problems, the performance of GH and LS has decreased substantially. The maximum of average deviations is 50.60% for GH and 15.58% for LS. If we omit 15.58%, LS has found solutions deviating not more than 7.36% of the best solution in average and the average percent error for all problems is 2.29%.

Table 7. Summary of computational results for VRP Instances.

					% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*			*p*	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil22	4	54	18	4,6,8	2.05	2.84	0.00	3.34	10.00	0.27	1195	1	3
	2	60	20	4,6,8,10	5.50	2.40	0.00	4.11	12.00	0.23	5103	0	2
	1	63	21	4,6,8,10	1.53	0.00	0.00	4.17	19.25	0.29	6990	0	2
eil23	5	54	18	4,6,8	0.05	0.04	0.00	3.45	12.00	0.23	1234	1	4
	2	63	21	4,6,8,10	0.87	0.00	0.00	4.39	20.00	0.29	7680	0	2
	1	66	22	4,6,8,10	50.60	15.58	0.00	4.45	19.25	0.22	11551	0	1
eil30	6	72	24	4,8,12	0.28	0.00	0.00	6.93	24.00	0.38	27202	0	1
	3	81	27	4,8,12	1.88	0.00	0.00	6.84	42.00	0.36	752	0	1
	1	87	29	4,8,12,14	12.44	7.36	0.00	8.66	44.75	0.38	1084	0	1
eil33	7	78	26	4,8,12	2.85	0.00	0.00	8.32	39.00	0.40	814	0	1
	3	90	30	4,8,12,14	0.37	4.76	0.00	8.99	47.50	0.60	1949	0	0
	1	96	32	4,8,12,16	15.44	1.75	0.00	9.85	54.50	0.47	3496	0	0
att48	10	114	38	5,10,15	1.85	0.61	0.00	12.16	81.33	1.19	28821	0	0
	5	129	43	5,10,15,20	4.62	0.39	0.00	20.68	101.75	1.12	50596	0	0
	1	141	47	5,10,15,20	18.65	3.06	0.00	20.15	163.50	1.11	76593	0	0
eil76	15	183	61	10,15,20,25	1.66	1.10	0.00	34.29	394.50	4.21		0	0
	8	204	68	10,15,20,25	0.14	0.69	0.00	37.10	519.25	8.09		0	0
	1	225	75	10,15,20,25	5.26	2.18	0.00	49.40	439.75	8.16		0	0
eil101	20	243	81	10,15,20,25	2.17	2.89	0.00	47.83	658.75	12.25		0	0
	10	273	91	10,15,20,25	2.16	1.69	0.00	51.56	827.00	4.91		0	0
	1	300	100	10,15,20,25	3.94	0.83	0.00	68.22	819.50	30.89		0	0
				Average	6.40	2.29	0.00	19.76	207.12	3.62	15004	0	1
				Maximum	50.60	15.58	0.00	68.22	827.00	30.89	76593	1	4

The reason for worse solutions may be the higher complexity generated by the higher number of population zones. In TSP instances each population zone is considered as an alternative collection point, while in VRP instances the number of population zones is three times the number of alternative collection points and they do not necessarily coincide. The maximum of percentage of TSPs solved is less than 4% for LS and less than 1% for GH. This is because of the larger solution space for problems generated from VRP instances. Even if the original problems have the same number of nodes, there are higher number of alternative sites in Type 2 problem sets which makes the solution space larger.

More detailed results can be found in Appendix E for VRP instances. A total of 78 problems have been solved and VNS has found the best solutions of all. While GH has found 17 of the best solutions, it is 45 for LS. The worst performance of LS and GH has been encountered in the third problem instance of eil23, see Table E2. In the case with 8 bottle banks, the deviation of the solution of GH is 76.16%. In the case with 6 bottle banks, the deviation of the solution of LS is 62.31%.

5.5.3 Results for Random Instances

The number of problems is rather small compared to Type 1 and Type 2 problem sets in random instances. Summary of computational results for random instances is given in Table 8. Once again, % Error is low like in the Type 1 instances. While VNS has solved all the problems optimally, average error is 0.96 % for GH and 0.39 % for LS. The reason for good results for GH and LS may be the location of alternative and compulsory sites at population zones. The relation between the average CPU times is similar to previous ones as VNS runs the fastest and LS the slowest. GH is around 12 times faster than LS while VNS runs 4 times faster than GH. In Appendix F, the details of the computational runs are summarized for Random Instances.

Table 8. Summary of computational results for Random Instances.

					% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*			*p*	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
ran20	10	60	20	4,6,8,10	0.12	0.60	0.00	4.11	15.00	0.49	5123.25	0.50	1.81
ran30	15	90	30	4,8,12,15	0.47	0.11	0.00	8.77	53.75	0.36	1732.00	0.11	0.39
ran40	20	120	40	5,10,15,20	0.40	0.01	0.00	15.47	118.25	4.12	38348.00	0.01	0.04
ran50	25	150	50	5,10,15,20	0.53	0.50	0.00	24.49	207.00	1.82	206371.00	0.00	0.02
ran60	30	180	60	5,10,15,20	2.64	0.90	0.00	26.53	281.50	1.12		0.00	0.01
ran80	40	240	80	5,10,15,20	0.11	0.00	0.00	40.22	534.25	6.18		0.00	0.00
ran100	50	300	100	5,10,15,20	2.46	0.58	0.00	60.52	959.50	32.99		0.00	0.00
				Average	0.96	0.39	0.00	25.73	309.89	6.72	62894	0	0
				Maximum	2.64	0.90	0.00	60.52	959.50	32.99	206371	0	2

LS has found 18 out of 28 of the best solutions while GH found 12 of them. VNS has found all the best solutions. The maximum percent deviation from the best solution is 5.11 for GH for the solution of ran100 with 20 bottle banks and 3.60 for LS for the solution of ran60 with 5 bottle banks.

5.5.4 Results for Large Instances

For the solution of large instances in addition to the algorithms we have already used, we have also utilized the RVNS algorithm in which the local search routine is omitted. We used two different time limits for RVNS: run time of GH and run time of LS in order to compare their efficiency. Two problems have been addressed, gil262 from VRP instances and kroA200 from TSP instances. The detailed computational results are given in Appendix G. GH, LS next to RVNS in the first row represents the time limit of the RVNS algorithm. VNS has found the best solutions for all the problems. RVNS is outperformed by LS and GH since it has higher percent error when run with the same time limit. The percent error of RVNS is as high as 13.14 when run with the time limit of GH. GH has found 4 of the best solutions out of 32, while LS has found 17. The maximum percent error is 4.37 for GH and 2.45 for LS and average error is 2.95 and 0.84 for gil262. The average error is even lower for kroA200, so these heuristics also produce acceptable results. LS algorithm runs far more slower than the other two, the maximum, more than 3 hours for one of gil262 instances. While VNS has run faster in kroA200 instances on average, it was slower for gil262 instances.

5.5.5 A detailed study on profits

Use of cheapest insertion heuristic instead of Linkern has substantially reduced the computation time. This has let us evaluate the results of locating different number of bottle banks. As the number of bottle banks to locate increased, the total number of possible combinations of locating them increases very rapidly and reaches its peak when the number of bottle banks to locate is half of the number of alternative sites. Hopefully, we could have run the algorithm for all possible *p*

values despite the complexity of the problem. In Table 9, the results for the random instance ran20 are summarized. Revenue(M) is the revenue obtained from compulsory collection points, which is a fixed amount. Revenue(K) is the revenue gained from alternative sites through bottle banks. Observe that in this problem instance the company has a loss when there is no bottle banks located and it has to locate at least 2 bottle banks in order to make profits.

Table 9. Revenues and costs for all possible p values for the problem ran20.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	-24.85	127.70	0.00	152.55
1	-0.44	127.70	24.61	152.75
2	18.05	127.70	43.15	152.80
3	36.35	127.70	61.45	152.80
4	50.54	127.70	76.24	153.40
5	63.71	127.70	93.76	157.75
6	76.23	127.70	106.33	157.80
7	85.74	127.70	116.09	158.05
8	88.97	127.70	119.02	157.75
9	91.92	127.70	122.57	158.35
10	94.20	127.70	125.20	158.70
11	94.61	127.70	132.36	165.45
12	94.26	127.70	132.36	165.80
13	93.15	127.70	136.35	170.90
14	92.70	127.70	136.35	171.35
15	90.16	127.70	136.96	174.50
16	88.31	127.70	136.96	176.35
17	86.05	127.70	138.00	179.65
18	83.20	127.70	138.65	183.15
19	78.65	127.70	139.05	188.10
20	71.82	127.70	151.97	207.85

In Figure 14, one can see the change on revenues and costs with increasing number of bottle banks. As expected, the revenue from bottle banks increases rapidly with increasing number of bottle banks in the beginning. The reason is that, few of the population zones are covered because of inadequate number of bottle banks. Most of the population zones are covered with 11 bottle banks and addition of new bottle banks does not make much contribution to the profit. Best

number of bottle banks to locate is 11 which produce a profit of 94.61. If we locate additional bottle banks, the profits decrease. The reason is that the increase in transportation costs is higher than the increase in the revenues. When most of the alternative sites are occupied, there are few empty sites left, most probably the ones far away from the current tour. Then, locating additional bottle banks causes higher increase in the transportation costs with inadequate increase in the revenues.

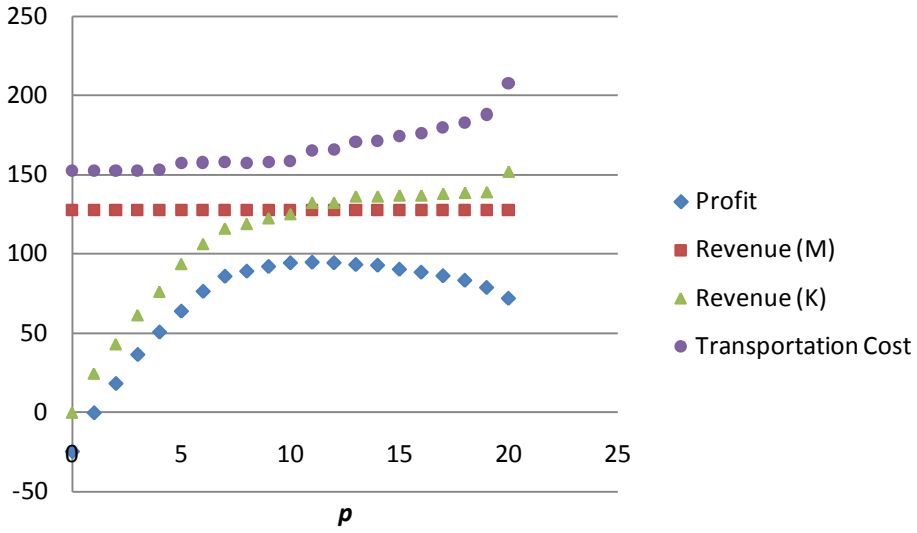

Figure 14. Revenue and cost components for the problem ran20.

When we have defined our problem, we have assumed that there is no cost of locating a bottle bank, except the tranportation cost. In case there is a fixed cost for each bottle bank located, we have calculated the optimum number of bottle banks to locate and the revenue obtained for different amount of unit cost. For a given fixed cost, the unit cost should be calculated by converting the fixed cost to daily amounts. In Figure 15, the best number of bottle banks to locate is given for the change in unit costs. When there is no unit cost, as expected, it is the best to locate 11 bottle banks which will produce the profit of 94.61. As the amount of unit cost increases, achieving high number of bottle banks become more costly, so it is more profitable to keep less number of bottle banks. Eventually, location of

any bottle banks will not make any contribution to the profit and it is better to operate only with the compulsory collection centers.

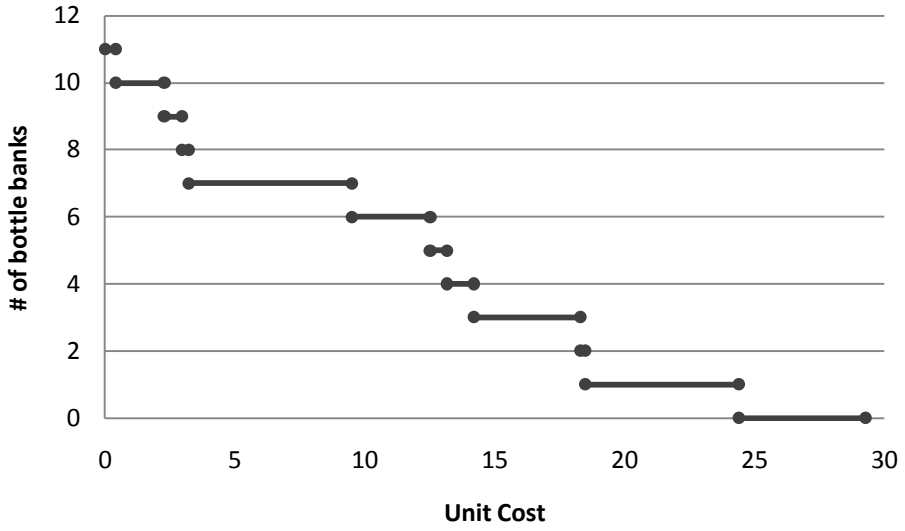

Figure 15. The best number of bottle banks to locate for the problem ran20.

In Figure 16, the profit obtained when the best number of bottle banks are located for a given unit cost is presented. As expected, when there is 0 unit cost, it is possible to earn the maximum profit of 94.61 by locating 11 bottle banks. Profits decrease with increasing unit cost, and eventually since it is not profitable to locate any bottle banks, only profits from compulsory centers are realized. In the Appendix H, similar results have been given for the randomly generated problems, ran50 and 100, TSP problem instances bayg29 and eil51, VRP problem instances att48 and eil76, and finally one of the large instances, kroA200.

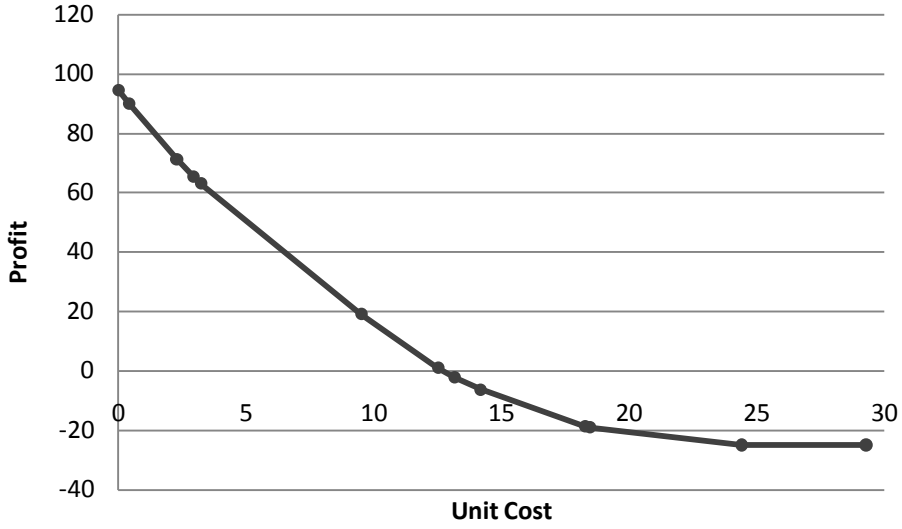

Figure 16. Amount of profit for different unit cost, ran20.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

In this thesis, a problem that was faced by a recycling company in Ankara in the collection of used glass is addressed. ÇEVKO, a non-governmental organization established to arrange recycling activities, has located a number of bottle banks to a number of districts according to the proximity to both residential zones and the recycling industry. Recently, ÇEVKO has handed over the possession of the bottle banks to a collecting company. However, with the given location of bottle banks, the company incurs loss. There are two sources of revenue for the company. The first one is a group of customers that it has made contract to collect glass in a regular basis, mostly hospitals, restaurants, bars and schools, which are defined as compulsory collection centers. The other source of revenue is the located bottle banks. The residents in the population zones drop their used glass to the nearby bottle banks and their willingness to return the glass depends on the closeness to the bottle bank. It is assumed that all the population is covered within some distance of a bottle bank. If this limit is exceeded, there is partial coverage of the population, and there is no coverage beyond the higher limit. The main cost of the company is the transportation cost which is incurred in the daily activity of collecting the recycled glass from both bottle banks and compulsory collection centers. Now, in order to increase its profits, the company has to locate a given number of bottle banks to some alternative collection points and find a new route for its collecting vehicle. The closeness to the potential population zones and the distance of the tour should be considered together so as the profit is maximized.

The problem has been modeled as a combination of two well known problems in the literature, the selective traveling salesman problem and maximal coverage location problem in the presence of partial coverage. The problem was handled by solving two nested problems: location of bottle banks and routing of a vehicle

through these locations and compulsory centers. Mainly three different methods have been used for the solution of the problem. Since it is a relatively new Metaheuristic approach and has given very good results for a number of location problems, we have used variable neighborhood search (VNS) algorithm by Mladenović and Hansen (1997) for the solution of location part of the problem. The second method is a greedy heuristic (GH) that locates one bottle bank at a time considering its contribution to both revenues and costs. The third method is the local search (LS) algorithm itself, which is one of the steps of the VNS. In order to obtain the optimum solutions, exhaustive enumeration has also been utilized where applicable that evaluates all possible combination of locations.

Each time a given location of bottle banks is evaluated, we have to find the corresponding route of the collecting vehicle. In order to find the best routes possible, three state of the art TSP solvers have been used, which are, Linkern, Concorde and LKH, among which Concorde is the one that uses an exact solution procedure. These three algorithms have been compared on some of the problems that have been generated for our problem. It has been found out that Linkern is the most efficient among all three for our problems, it has been selected for the solution of TSPs generated by GH, LS and VNS. Concorde solver has also been utilized for the smallest problems where Linkern does not work, in addition to the exhaustive search procedure that requires optimum solutions to generated TSPs. Since during the execution of VNS algorithm there is a great number of TSPs to solve, the combination of VNS with Linkern becomes inefficient. In order to overcome this problem, we have tried the algorithms, cheapest insertion and 2-opt in the place of Linkern. These algorithms run faster but does not give as good results as Linkern. We have found out that, VNS combined with Linkern in shaking step and cheapest insertion in local search runs for some instances more than 1000 times faster than VNS combined with only Linkern. So we have adopted this last strategy.

In order to compare our algorithms, we have generated problems using the original TSP and VRP instances from TSPlib, in addition to the ones created randomly. The problems differ from each other with respect to the ratio between the number of compulsory collection centers, alternative sites and population zones. For each of TSP and VRP instance 3 different problems have been generated. Both considering the solution quality and CPU time, VNS-L-I is the best of the three algorithms. It has found all the best solutions and all the known optimum solutions. Reduced VNS, which is a variant of VNS that was developed for solving large problems in shorter time, has also been used for our two large instances. However, VNS was more successful, as once again finding all the best results, better than RVNS, in comparable CPU time.

Since our algorithm became very efficient with the use of cheapest insertion heuristic, it was possible to solve the problems for all possible p values. This way we were able to find the best number of bottle banks to locate to maximize the profit of the company. More important than that, in case there is a fixed cost of a bottle bank, we could calculate the best number of bottle banks to locate and the resulting profit. Some of the curves for the best number of bottle banks to locate and the profits versus unit cost have also been given.

As a future work, one can study exact solution approaches to find optimum results. In addition to that, by assuming that the company has a freedom of deciding the number of bottle banks to locate, one can address the problem of finding the optimum number of bottle banks, their location and routing, which is a similar problem to selective TSP. It is also possible to study the effect of the change of unit transportation cost to the location of bottle banks and the overall profits since it is considered to be the main source of costs and fuel prices are increasing continuously. For large problems, multivehicle version of the same problem can be addressed with the inclusion of capacity and time limits, since one vehicle will not be sufficient to serve every collection point in a daily basis.

REFERENCES

1. Applegate D., Bixby R., Chvatal V., Cook W. (2000), "TSP cuts which do not conform to the template paradigm", Lecture Notes In Computer Science, Volume 2241, Pages 261-304

2. Applegate D., Cook W., Rohe A. (2003), "Chained Lin-Kernighan for large traveling salesman problems", INFORMS Journal on Computing, Volume 15, Issue 1, Pages 82-92

3. Arakaki R.G.I., Lorena L.A.N. (2001), "A constructive genetic algorithm for the maximal covering location problem", Proceedings of Metaheuristics International Conference, Porto, Portugal

4. Batta R., Mannur N.R. (1990), "Covering-location models for emergency situations that require multiple response units", Management Science, Volume 36, No 1, Pages 16-23

5. Berman O., Krass D. (2002), "The generalized maximal covering location problem", Computers & operations research, Volume 29, Issue 6, Pages 563–581

6. Blum C., Roli A. (2003), "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", ACM Computing Surveys (CSUR) Archive, Volume 35 , Issue 3, Pages 268 – 308

7. Braysy, O. (2003), "A reactive variable neighborhood search for the vehicle-routing problem with time windows", INFORMS Journal on Computing, Volume 15, Issue 4, Pages 347-368

8. Burke E.K., Cowling P.I., Keuthen R. (2001), "Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem", Lecture Notes in Computer Science, Volume 2037, Pages 203-212

9. Carrabs F., Cordeau J. F., Laporte G. (2007), "Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading", INFORMS Journal on Computing, Volume 19, Issue 4, Pages 618-648

10. Chao I.M., Golden B.L., Wasil E.A. (1996), "A fast and effective heuristic for the orienteering problem", European Journal of Operational Research, Volume 88, Pages 475-489

11. Chung C. (1986), "Recent applications of the maximal covering location planning (M.C.L.P.) model", The Journal of the Operational Research Society, Volume 37, No. 8, Pages 735-746

12. Church R., ReVelle C. (1974), "The maximal covering location problem", Papers of the Regional Science Association, Volume 32, Pages 101-118

13. Church R.L., Roberts K.L. (1983), "Generalized coverage models and public facility location", Papers in Regional Science, Volume 53, Pages 117-135

14. Crainic T.G., Gendreau M., Hansen P., Mladenović N. (2004), "Cooperative parallel variable neighborhood search for the p-median", Journal of Heuristics, Volume 10, Number 3, Pages 293-314

15. Croes G. A. (1958), "A method for solving traveling salesman problems", Operations Research, Volume 6, Pages 791-812

16. DellAmico M., Maffioli F., Varbrand P. (1995), "On prize-collecting tours and the asymmetric travelling salesman problem", International Transactions in Operational Research, Volume 2, Issue 3, Pages 297-308

17. Feillet D., Dejax P., Gendreau M. (2005), "Traveling salesman problems with profits", Transportation Science, Volume 39, Issue 2, Pages 188-205

18. Fischetti M., Gonzalez J.J. S., Toth P. (1998), "Solving the orienteering problem through branch-and-cut", INFORMS Journal on Computing, Volume 10, No 2, Pages 133-148

19. Fleszar K., Osman I.H., Hindi K.S. (2009) "A variable neighbourhood search algorithm for the open vehicle routing problem", European Journal of Operational Research, Volume 195, Issue 3, Pages 803–809

20. Galvao R.D., Espejo L.G.A., Boffey B. (2000), "A comparison of Lagrangean and surrogate relaxations for the maximal covering location problem", European Journal of Operational Research, Volume 124, Issue 2, Pages 377-389

21. García-López F., Melián-Batista B., Moreno-Pérez J.A. (2002), "The parallel variable neighborhood search for the p-median problem", Journal of Heuristics, Volume 8, Number 3, Pages 375-388

22. Gendreau M., Laporte G., Semet F. (1998), "A branch-and-cut algorithm for the undirected selective traveling salesman problem", Networks, Volume 32, Issue 4, Pages 263-273

23. Gendreau M., Laporte G., Semet F. (1998), "A tabu search heuristic for the undirected selective travelling salesman problem", European Journal of Operational Research, Volume 106, Pages 539-545

24. Gribkovskaia I., Laporte G., Shyshou A. (2008), "The single vehicle routing problem with deliveries and selective pickups", Computers & Operations Research, Volume 35, Issue 9, Pages 2908-2924

25. Gutierrez-Jarpa, G., Marianov V., Obreque C. (2009), "A single vehicle routing problem with fixed delivery and optional collections ", IIE Transactions, Volume 41, Issue 12, pages 1067-1079

26. Hansen P., Brimberg J., Urosević D., Mladenović N. (2007), "Primal-dual variable neighborhood search for the simple plant-location problem", INFORMS Journal on Computing, Volume 19, Issue 4, Pages 552-564

27. Hansen P., Mladenović N. (1997), "Variable neighborhood search for the p-median", Location Science, Volume 5, Issue 4, Pages 207-226

28. Hansen P., Mladenović N. (2001), "Variable neighborhood search: Principles and applications", European Journal of Operational Research, Volume 130, Issue 3, Pages 449-467

29. Hansen P., Mladenović N., Perez-Britos D. (2001), "Variable neighborhood decomposition search", Journal of Heuristics, Volume 7, Issue 4, Pages 335-350

30. Harm G., Hentenryck P. V. (2005), "A multistart variable neighborhood search for uncapacitated facility location", Proceedings of the 6th Metaheuristics International Conference, Vienna, Austria

31. Helsgaun, K. (2000), “An effective implementation of the Lin–Kernighan traveling salesman heuristic”, European Journal of Operational Research, Volume 126, Issue , Pages 106-130

32. Hemmelmayr V.C., Doerner K.F., Hartl R.F. (2009), “A variable neighborhood search heuristic for periodic routing problems”, European Journal of Operational, Volume 195, Issue 3, Pages 791–802

33. Kantor M.G., Rosenwein M.B. (1992), “The orienteering problem with time windows”, The Journal of the Operational Research Society, Volume 43, No. 6, Pages 629- 635

34. Karasakal O., Karasakal E. K. (2004), “A maximal covering location model in the presence of partial coverage”, Computers & Operations Research, Volume 31, Issue 9, Pages 1515-1526

35. Kytöjoki J., Nuortio T., Bräysy O., Gendreau M. (2007), “An efficient variable neighborhood search heuristic for very large scale vehicle routing problems”, Computers and Operations Research, Volume 34, Issue 9, Pages 2743 – 2757

36. Laporte G., Martello S. (1990), “The selective travelling salesman problem”, Discrete Applied Mathematics, Volume 26, Issue 2-3, Pages 193-207

37. Liang Y.C., Kulturel-Konak S., Smith A.E. (2002), “Meta heuristics for the orienteering problem”, Evolutionary Computation, Volume 1, Pages 384-389

38. Liang Y.C., Smith A.E. (2006), “An ant colony approach to the orienteering problem”, Journal of the Chinese Institute of Industrial Engineers, Volume 23, Pages 403-414

39. Lin S., Kernighan B.W. (1973), “An effective heuristic algorithm for the traveling-salesman problem”, Operations Research, Volume 21, Issue 2, Pages 498-516

40. Lorena L.A.N., Pereira M.A. (2002), “A Lagrangean/surrogate heuristic for the maximal covering location problem using Hillsman’s edition”, International Journal of Industrial Engineering, Volume 9, Issue 1, Pages 57-67

41. Millar H.H., Kiragu M. (1997), "A time-based formulation and upper bounding scheme for the selective travelling salesperson problem", The Journal of the Operational Research Society, Volume 48, Issue 5, Pages 511-518

42. Miller C.E., Tucker A.W., Zemlin R.A. (1960), "Integer programming formulation of traveling salesman problems", Journal of the ACM (JACM), Volume 7, Issue 4, Pages 326 - 329

43. Mladenović N., Dražić M., Kovačevic-Vujčić V., Čangalović M. (2008), "General variable neighborhood search for the continuous optimization", European Journal of Operational Research, Volume 191, Issue 3, Pages 753-770

44. Mladenović N., Hansen P. (1997), "Variable neighborhood search", Computers & Operations Research, Volume 24, Issue 11, Pages 1097-1100

45. Mladenović N., Labbé M., Hansen P. (2003), "Solving the p-center problem with tabu search and variable Neighborhood Search", Networks, Volume 42, Issue 1, Pages 48–64

46. Mladenović N., Moreno J.P., Moreno-Vega J. (1996), "A chain-interchange heuristic method", Yugoslav Journal of Operational Research, Volume 6, Issue 1, Pages 41-54

47. Paraskevopoulos D.C., Repoussis P.P., Tarantilis C.D., Ioannou G., Prastacos G.P. (2008), "A reactive variable neighborhood tabu search for the heterogeneous fleet vehicle routing problem with time windows", Journal of Heuristics, Volume 14, Issue 5, Pages 425–455

48. Pereira M.A., Lorena L., Senne E. (2007), "A column generation approach for the maximal covering location problem" , International Transactions in Operational Research, Volume 14, Issue 4, Pages 349-364

49. Pérez J.A.M., Hansen P., Mladenovic N. (2004), "Parallel variable neighborhood search", Les Cahiers du GERAD, Group for Research in Decision Analysis, Montréal, Canada , Reference no: G–2004–92

50. Pirkul H., Schilling D. (1989), "The capacitated maximal covering location problem with backup service", Annals of Operations Research, Volume 18, Issue 1, Pages 141-154

51. Pirkul H., Schilling D. (1991), "The maximal covering location problem with capacities on total workload", Management Science, Volume 37, Issue 2, Pages 233-248

52. Polacek M., Benkner S., Doerner K. F., Hartl R. F (2008), "A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows", Official Open Access Journal of VHB, Volume 1, Issue 2, Pages 207—218

53. Polacek M., Hartl R.F., Doerner K., Reimann M. (2004), "A variable neighborhood search for the multi depot vehicle routing problem with time windows", Journal of Heuristics, Volume 10, Issue 6, Pages 613–627

54. Polat E. (2008), "A location and routing-with-profit problem in glass recycling", Thesis (M.S.), Middle East Technical University

55. Puchinger J. Raidl G. R. (2008), "Bringing order into the neighborhoods: relaxation guided variable neighborhood search", Journal of Heuristics, Volume 14, Issue 5, Pages 457–472

56. Resende M.G.C. (1998), "Computing approximate solutions of the maximum covering problem with GRASP", Journal of Heuristics, Volume 4, Issue 2, Pages 161-177

57. Righini G., Salani M. (2006), "Dynamic programming for the orienteering problem with time windows", Technical report, Department of Information Technology, The University of Milan, Crema, Italy

58. Rosenkrantz D.J., Stearns R.E., Lewis M. (1977), "Approximate algorithms for the traveling salesperson problem", SIAM Journal on Computing, Volume 6, Issue 3

59. Schilling D.A., Jayaraman V., Barkhi R. (1993), "A review of covering problems in facility location", Location Science, Volume 1, Issue 1, Pages 25-55

60. Sevkli M., Aydin M.E. (2007), “Parallel variable neighbourhood search algorithms for job shop scheduling problems”, IMA Journal of Management Mathematics, Volume 18, Pages 117−133

61. Sevkli Z., Sevilgen F. E. (2006), “Variable Neighborhood Search for the Orienteering Problem”, Lecture notes in computer science, Volume 4263, Pages 134-143

62. Süral H., Bookbinder J.H. (2003), “The single-vehicle routing problem with unrestricted backhauls”, Networks, Volume 41 Issue 3, Pages 127-136

63. Tasgetiren M.F., Smith A.E. (2000), “A genetic algorithm for the orienteering problem”, Evolutionary Computation, Volume 2, Pages 910-915

64. Tsiligirides T. (1984), “Heuristic methods applied to orienteering”, The Journal of the Operational Research Society, Volume 35, Issue 9, Pages 797-809

65. Vansteenwegena P., Souffriaua W., Van Oudheusdena D. (2010), “The orienteering problem: A survey”, European Journal of Operational Research, Article in Press

66. Wang Q., Sun X., Golden B.L., Jia J. (1995), “Using artificial neural networks to solve the orienteering problem”, Annals of Operations Research, Volume 61, Pages 111-120

67. Wen M., Krapper E., Larsen J., Stidsen T. K. (2009), “A multi-level variable neighborhood search heuristic for a practical vehicle routing and driver scheduling problem”, DTU Management 2009, pages 1-31

68. Yazdani M., Amiri M., Zandieh M. (2010), “Flexible job-shop scheduling with parallel variable neighborhood search algorithm”, Expert Systems with Applications, Volume 37, Pages 678–687

APPENDIX A

STEPS OF VND, VNDS AND SVNS

Steps of the basic VND:

Initialization. Select the set of neighborhood structures $\mathcal{N}'_k$, $k = 1, \ldots, k'_{max}$, that will be used in the descent; find an initial solution x;

Repeat the following until no improvement is obtained:
(1) Set $k \leftarrow 1$; (2) Until $k = k'_{max}$, repeat the following steps:
(a) *Exploration of neighborhood.* Find the best neighbor x' of x ($x' \in \mathcal{N}'_k(x)$);
(b) *Move or not.* If the solution thus obtained x' is better than x, set $x \leftarrow x'$; otherwise, set $k \leftarrow k + 1$.

Steps of the basic VNDS:

Initialization. Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{max}$, that will be used in the search; find initial solution x; choose stopping condition;

Repeat the following until the stopping condition is met:
(1) Set $k \leftarrow 1$; (2) Until $k = k_{max}$, repeat the following steps:
(a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$); in other words, let y be a set of k solution attributes present in x' but not in x ($y = x' \setminus x$).
(b) *Local search.* Find the local optimum in the space of y either by inspection or by some heuristic; denote the best solution found with y' and with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$);
(c) *Move or not.* If the solution thus obtained is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with $\mathcal{N}_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;

Steps of the SVNS:

Initialization. Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{max}$, that will be used in the search; find an initial solution x and its value $f(x)$; set $x_{opt} \leftarrow x$, $f_{opt} \leftarrow f(x)$: choose a stopping condition and a parameter value α;

Repeat the following until the stopping condition is met:
(1) Set $k \leftarrow 1$; (2) Until $k = k_{max}$, repeat the following steps:
(a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$);
(b) *Local search.* Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
(c) *Improve or not.* If $f(x'') < f_{opt}$ set $f_{opt} \leftarrow f(x)$ and $x_{opt} \leftarrow x''$;
(d) *Move or not.* If $f(x'') - \alpha\rho(x, x'') < f(x)$ set $x \leftarrow x''$ and $k \leftarrow 1$; otherwise set $k \leftarrow k + 1$.

APPENDIX B

COMPARISON OF CONCORDE, LINKERN AND LKH ALGORITHMS

Table B-1. CPU Time and % Error of TSP Algorithms for TSP instances.

		CPU (seconds)			% Error	
Name	TSP length	Concorde	Linkern	LKH	Linkern	LKH
burma14	14	0.063	0.031	0.046	0	0
ulysses16	16	0.093	0.046	0.078	0	0
ulysses22	22	0.109	0.078	0.078	0	0
bayg29	29	0.187	0.046	0.062	0	0
dantzig42	42	0.108	0.046	0.062	0	0
att48	48	0.141	0.078	0.093	0	0
eil51	51	0.326	0.062	0.062	0	0
st70	70	0.14	0.078	0.093	0	0
pr76	76	0.421	0.141	0.156	0	0
eil101	101	0.234	0.093	0.094	0	0
pr124	124	0.655	0.171	0.171	0	0
ch150	150	0.684	0.125	0.171	0.18	0.05
kroA200	200	0.999	0.203	0.375	0	0

Table B-2. CPU Time and % Error of TSP Algorithms for VRP instances.

		CPU (seconds)			% Error	
Name	TSP length	Concorde	Linkern	LKH	Linkern	LKH
eil22	22	0.078	0.031	0.062	0	0
eil23	23	0.063	0.046	0.078	0	0
eil30	30	0.093	0.078	0.078	0	0
eil33	33	0.14	0.062	0.078	0	0
att48	48	0.187	0.063	0.093	0	0
eil51	51	0.312	0.047	0.078	0	0
eilA76	76	0.124	0.063	0.078	0	0
eilA101	101	0.141	0.078	0.094	0	0
gil262	262	1.406	0.234	0.625	0	0

Table B-3. CPU Time and % Error of TSP Algorithms for TSPs generated fromVRP instances.

							Average CPU (seconds)			# of times not optimal		Average % Error	
Name	# of cities	*m*	*s*	*n*	*p*	TSP length	Concorde	Linkern	LKH	Linkern	LKH	Linkern	LKH
eil22	22	1	21	63	10	11	0.046	0.034	0.072	0	0	0	0
		2	20	60	10	12	0.047	0.034	0.072	0	0	0	0
		4	18	54	8	12	0.048	0.033	0.067	0	0	0	0
eil23	23	1	22	66	10	11	0.046	0.035	0.068	0	0	0	0
		2	21	63	10	12	0.047	0.034	0.068	0	0	0	0
		5	18	54	8	13	0.047	0.035	0.068	0	0	0	0
eil30	30	1	29	87	14	15	0.055	0.037	0.068	0	0	0	0
		3	27	81	12	15	0.054	0.038	0.068	0	0	0	0
		6	24	72	12	18	0.061	0.042	0.068	2	0	0.0084	0
eil33	33	1	32	96	16	17	0.058	0.038	0.068	0	0	0	0
		3	30	90	14	17	0.057	0.037	0.068	0	0	0	0
		7	26	78	12	19	0.063	0.04	0.068	0	0	0	0
att48	48	1	47	141	20	21	0.064	0.041	0.068	0	0	0	0
		5	43	129	20	25	0.067	0.042	0.068	0	0	0	0
		10	38	114	15	25	0.069	0.044	0.068	0	0	0	0

Table B-3. CPU Time and % Error of TSP Algorithms for TSPs generated fromVRP instances. (cont'd).

							Average CPU (seconds)			# of times not optimal		Average % Error	
Name	# of cities	*m*	*s*	*n*	*p*	TSP length	Concorde	Linkern	LKH	Linkern	LKH	Linkern	LKH
eil51	51	1	50	150	25	26	0.062	0.038	0.068	0	0	0	0
		5	46	138	20	25	0.061	0.037	0.068	0	0	0	0
		10	41	123	20	30	0.067	0.039	0.068	1	0	0.0016	0
eilA76	76	1	75	225	25	26	0.12	0.038	0.068	0	0	0	0
		8	68	204	25	33	0.081	0.042	0.068	0	0	0	0
		15	61	183	25	40	0.111	0.045	0.068	0	1	0	0.0014
eilA101	101	1	100	300	25	26	0.064	0.038	0.068	0	1	0	0.0008
		10	91	273	25	35	0.091	0.043	0.068	0	0	0	0
		20	81	243	25	45	0.146	0.048	0.071	2	3	0.002	0.0049
gil262	262	1	261	783	20	21	0.055	0.036	0.067	0	0	0	0
		26	236	708	20	46	0.113	0.053	0.068	1	2	0.0018	0.0011
		52	210	630	20	72	0.312	0.068	0.099	1	7	0.0021	0.0096
									Total	7	14		

APPENDIX C

COMPARISON OF THE PERFORMANCE OF VNS WITH ALTERNATIVE TSP SOLUTION PROCEDURES

Table C-1. CPU Time and % Error of VNS with alternative TSP solution procedures for TSP instance eil101.

			% Deviation from Best				CPU (seconds)			
m-n-s	*p*	f(best)	VNS+L	VNS-L-I	VNS-I-I	VNS-I-2-I	VNS+L	VNS-L-I	VNS-I-I	VNS-I-2-I
101-20-20	4	731.69	0.00	0.00	0.85	0.37	98	0.64	0.09	0.34
	6	739.84	0.00	0.00	0.62	0.21	273	0.95	0.11	0.11
	8	742.19	0.00	0.00	0.16	0.86	429	1.08	0.11	0.11
	10	740.63	0.00	0.00	0.00	0.62	701	1.48	0.13	0.11
101-40-40	4	767.94	0.00	0.00	0.12	0.62	394	0.72	0.11	0.17
	8	794.42	0.00	0.00	0.28	0.00	1241	0.97	0.13	0.19
	12	805.03	0.00	0.00	1.34	0.00	3551	6.45	0.30	0.25
	16	812.89	0.00	0.00	0.37	0.82	6490	8.69	0.39	0.53
	20	814.20	0.00	0.00	2.12	1.38	9889	5.45	0.69	0.80
101-60-60	4	843.63	0.00	0.00	1.46	0.52	387	0.59	0.13	0.11
	8	886.40	0.00	0.00	0.76	0.00	2064	1.19	0.23	0.20
	12	910.85	0.00	0.00	1.23	0.84	6976	1.81	0.44	0.92
	16	921.43	0.00	0.00	0.85	1.34	22905	2.99	1.03	0.92
	20	922.68	0.00	0.00	2.13	0.46	22529	7.20	2.03	1.75
101-80-80	4	932.04	0.00	0.00	0.15	0.45	512	0.69	0.13	0.13
	8	990.69	0.00	0.04	0.05	0.73	3342	1.95	0.45	0.41
	12	1023.00	0.02	0.00	0.29	0.42	14859	8.22	1.59	1.06
	16	1041.57	0.00	0.03	0.01	0.28	19185	5.39	1.92	3.17
		Average	0.00	0.00	0.71	0.55	6434.72	3.14	0.56	0.63
		Maximum	0.02	0.04	2.13	1.38	22905.00	8.69	2.03	3.17
		# Best	17	16	1	2				
		Total	18							

Table C-2. CPU Time and % Error of VNS with alternative TSP solution procedures for VRP instance eil101.

			% Deviation from Best				CPU (seconds)			
m-n-s	*p*	f(best)	VNS-L	VNS-L-I	VNS-I-I	VNS-I-2-I	VNS-L	VNS-L-I	VNS-I-I	VNS-I-2-I
20 243 81	10	487.89	0.00	0.00	2.27	0.00	1949	4.47	2.03	1.06
	15	576.53	0.00	0.00	6.00	1.37	3980	4.81	3.97	16.26
	20	622.36	0.00	0.00	4.24	0.42	3394	10.14	52.73	11.33
	25	642.79	0.00	0.00	1.91	1.56	8451	77.20	18.88	292.48
10 273 91	10	457.63	0.00	0.00	3.88	0.00	1668	1.95	2.09	1.42
	15	574.93	0.00	0.00	4.17	2.08	3019	9.52	7.56	23.28
	20	623.46	0.00	0.00	1.58	0.00	4894	15.06	40.64	33.83
	25	640.74	0.00	0.00	3.63	0.24	21647	46.42	113.15	34.94
1 300 100	10	384.52	0.00	0.00	0.00	0.00	1430	9.95	2.84	2.86
	15	488.06	0.00	0.00	0.00	0.00	2974	9.81	6.56	21.89
	20	550.19	0.00	0.00	0.00	0.00	11296	20.03	43.12	16.49
	25	576.64	0.00	0.00	0.00	0.00	7868	172.64	69.88	125.37
		Average	0.00	0.00	2.31	0.47	6047.50	31.83	30.29	48.43
		Maximum	0.00	0.00	6.00	2.08	21647.00	172.64	113.15	292.48
		# Best	12	12	4	7				
		Total	12							

APPENDIX D

COMPUTATIONAL RESULTS FOR TSP INSTANCES

In this appendix, computational results for all TSP instances that comprise burma14, bayg29, dantzig42, eil51 and eil101 are given in detail. All of the instances except eil101 are composed of 3 different problems and each problem has been solved for different number of bottle banks except the smallest problem, burma14. CPU time, percent deviation from the best solution and percentage of all possible TSPs solved are presented for each *p*. The results are given for VNS, Local Search and Greedy Heuristic algorithms and where applicable, solution results for Exhaustive Search are also provided.

Table D-1. Computational Results for the TSP instance, burma14.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
burma14	14-7-7	3	111.20	13.69	1	18	13.90	1	24	13.90	0.80	9	13.90	2	35
	14-14-14	4	113.00	25.42	2	50	27.11	8	200	27.11	0.17	1	27.11	57	1001
		5	113.00	27.03	2	60	28.52	14	360	28.52	0.22	1	28.52	116	2002
		6	113.00	28.44	3	69	29.71	11	288	29.71	0.52	2	29.71	187	3003
		7	113.00	27.83	3	77	25.84	7	196	29.40	0.31	1	29.40	214	3432
	14-21-21	4	95.90	52.10	3	78	55.87	8	204	55.87	0.44	2	55.87	359	5985
		6	95.90	59.05	4	111	59.05	14	360	60.07	0.23	1	60.07	3416	54264
		8	95.90	57.96	5	140	59.12	17	416	59.12	0.64	1	59.12	13262	203490
		10	95.90	55.74	6	165	56.27	32	770	56.60	0.97	4	56.60	23869	352716

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
burma14	14-7-7	3*	13.90	1.51	0.00	0.00	1	1	0.80	2	51.43	68.57
	14-14-14	4*	27.11	6.22	0.00	0.00	2	8	0.17	57	5.00	19.98
		5*	28.52	5.22	0.00	0.00	2	14	0.22	116	3.00	17.98
		6*	29.71	4.27	0.00	0.00	3	11	0.52	187	2.30	9.59
		7*	29.40	5.35	12.11	0.00	3	7	0.31	214	2.24	5.71
	14-21-21	4*	55.87	6.74	0.00	0.00	3	8	0.44	359	1.30	3.41
		6*	60.07	1.70	1.70	0.00	4	14	0.23	3416	0.20	0.66
		8*	59.12	1.96	0.00	0.00	5	17	0.64	13262	0.07	0.20
		10*	56.60	1.50	0.58	0.00	6	32	0.97	23869	0.05	0.22
			Average	3.83	1.60	0.00	3	12	0.48	4609	7.29	14.04
			Maximum	6.74	12.11	0.00	6	32	0.97	23869	51.43	68.57
			# Best	0	6	9						
			Total	9								

Table D-2. Computational Results for the TSP instance, bayg29.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
bayg29	29-15-15	4	220.20	25.36	2.33	54	25.36	8	176	25.36	0.41	2	25.36	139	1365
		5	220.20	26.53	2.80	65	26.53	9	200	26.53	0.36	1	26.53	310	3003
		6	220.20	27.66	3.23	75	27.66	15	324	27.66	0.97	3	27.66	516	5005
		7	220.20	27.61	3.61	84	27.61	10	224	27.61	0.69	2	27.61	671	6435
	29-29-29	4	226.50	44.73	4.66	110	44.73	17	400	44.73	0.25	1	44.73	2509	23751
		6	226.50	52.52	6.72	159	52.52	31	690	52.91	0.55	2	52.91	54701	475020
		8	226.50	57.76	8.70	204	59.37	69	1512	59.37	0.52	1			4292145
		10	226.50	59.52	10.77	245	60.82	79	1710	60.82	0.31	1			20030010
		12	226.50	59.98	12.45	282	60.70	75	1632	60.70	4.62	17			51895935
		14	226.50	59.64	14.06	315	55.80	63	1260	60.02	1.13	3			77558760
	29-45-45	4	213.50	65.24	7.58	174	65.24	38	820	65.24	0.61	3	65.24	14781	148995
		8	213.50	91.48	14.95	332	91.48	127	2664	91.48	1.47	3			2.16E+08
		12	213.50	98.67	22.28	474	99.09	246	4752	99.09	5.02	15			2.88E+10
		16	213.50	99.09	29.37	600	98.99	204	3712	99.70	0.42	1			6.47E+11
		20	213.50	97.27	35.67	710	97.96	438	8000	98.22	2.59	5			3.17E+12

Table D-2. Computational Results for the TSP instance, bayg29. (cont'd)

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
bayg29	29-15-15	4*	25.36	0.00	0.00	0.00	2	8	0.41	139	3.96	12.89
		5*	26.53	0.00	0.00	0.00	3	9	0.36	310	2.16	6.66
		6*	27.66	0.00	0.00	0.00	3	15	0.97	516	1.50	6.47
		7*	27.61	0.00	0.00	0.00	4	10	0.69	671	1.31	3.48
	29-29-29	4*	44.73	0.00	0.00	0.00	5	17	0.25	2509	0.46	1.68
		6*	52.91	0.75	0.75	0.00	7	31	0.55	54701	0.03	0.15
		8	59.37	2.71	0.00	0.00	9	69	0.52		0.00	0.04
		10	60.82	2.15	0.00	0.00	11	79	0.31		0.00	0.01
		12	60.70	1.19	0.00	0.00	12	75	4.62		0.00	0.00
		14	60.02	0.64	7.03	0.00	14	63	1.13		0.00	0.00
	29-45-45	4*	65.24	0.00	0.00	0.00	8	38	0.61	14781	0.12	0.55
		8	91.48	0.00	0.00	0.00	15	127	1.47		0.00	0.00
		12	99.09	0.42	0.00	0.00	22	246	5.02		0.00	0.00
		16	99.70	0.61	0.71	0.00	29	204	0.42		0.00	0.00
		20	98.22	0.97	0.27	0.00	36	438	2.59		0.00	0.00
* f(best) is the optimum.			Average	0.63	0.58	0.00	12	95	1.33	10518	0.64	2.13
			Maximum	2.71	7.03	0.00	36	438	5.02	54701	3.96	12.89
			# Best	7	11	15						
			Total	15								

Table D-3. Computational Results for the TSP instance, dantzig42.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
dantzig42	42-21-21	4	306.40	34.77	4	78	34.77	13	272	34.77	0.28	1	34.77	643	5985
		6	306.40	42.61	5	111	42.61	28	540	42.61	0.30	1	42.61	6287	54264
		8	306.40	47.96	7	140	46.23	32	624	47.96	0.88	1	47.96	26075	203490
		10	306.40	47.49	8	165	47.49	44	770	47.49	1.64	5	47.49	50842	352716
	42-42-42	4	327.50	49.18	8	162	49.18	36	760	49.18	0.27	1	49.18	10786	111930
		8	327.50	82.91	15	308	82.91	68	1360	82.91	0.50	1			1.18E+08
		12	327.50	101.37	21	438	99.90	130	2520	101.37	0.53	1			1.11E+10
		16	327.50	103.96	27	552	103.96	198	3744	103.96	0.47	1			1.67E+11
		20	327.50	102.42	32	650	102.42	273	4840	102.42	0.69	1			5.14E+11
	42-63-63	4	331.50	73.55	12	246	73.55	60	1180	73.55	0.48	2	73.55	69081	595665
		8	331.50	112.46	24	476	111.79	232	3960	112.71	2.59	5			3.87E+09
		12	331.50	127.87	36	690	129.72	337	6120	129.72	2.14	4			2.67E+12
		16	331.50	138.86	47	888	141.58	450	8272	141.58	0.58	1			3.66E+14
		20	331.50	142.43	57	1070	143.81	676	11180	144.67	1.17	1			1.35E+16

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
dantzig42	42-21-21	4*	34.77	0.00	0.00	0.00	4	13	0.28	643	1.30	4.54
		6*	42.61	0.00	0.00	0.00	5	28	0.30	6287	0.20	1.00
		8*	47.96	0.00	3.60	0.00	7	32	0.88	26075	0.07	0.31
		10*	47.49	0.00	0.00	0.00	8	44	1.64	50842	0.05	0.22
	42-42-42	4*	49.18	0.00	0.00	0.00	8	36	0.27	10786	0.14	0.68
		8	82.91	0.00	0.00	0.00	15	68	0.50		0.00	0.00
		12	101.37	0.00	1.45	0.00	21	130	0.53		0.00	0.00
		16	103.96	0.00	0.00	0.00	27	198	0.47		0.00	0.00
		20	102.42	0.00	0.00	0.00	32	273	0.69		0.00	0.00
	42-63-63	4*	73.55	0.00	0.00	0.00	12	60	0.48	69081	0.04	0.20
		8	112.71	0.22	0.82	0.00	24	232	2.59		0.00	0.00
		12	129.72	1.43	0.00	0.00	36	337	2.14		0.00	0.00
		16	141.58	1.92	0.00	0.00	47	450	0.58		0.00	0.00
		20	144.67	1.55	0.60	0.00	57	676	1.17		0.00	0.00
* f(best) is the optimum.			Average	0.37	0.46	0.00	22	184	0.89	27286	0.13	0.50
			Maximum	1.92	3.60	0.00	57	676	2.59	69081	1.30	4.54
			# Best	10	10	14						
			Total	14								

Table D-4. Computational Results for the TSP instance, eil51.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit	CPU (s)	Total # of TSPs
eil51	51-25-25	4	366.61	46.20	5	94	46.20	22	420	46.20	0.45	2	46.20	3285	12650
		6	366.61	58.89	7	135	55.96	31	570	58.89	0.44	1	58.89	54533	177100
		8	366.61	66.31	8	172	66.31	80	1360	66.31	0.73	1			1081575
		10	366.61	70.13	10	205	69.26	46	750	70.13	0.44	1			3268760
		12	366.61	69.82	12	234	69.56	66	1092	69.87	2.23	6			5200300
	51-51-51	4	400.81	53.42	10	198	53.42	40	752	53.42	0.45	2	53.42	71859.00	249900
		8	400.81	86.26	20	380	86.26	179	3440	86.26	0.80	1			6.37E+08
		12	400.81	106.64	30	546	106.30	301	5616	107.14	0.98	2			1.59E+11
		16	400.81	116.96	38	696	120.08	238	4480	120.08	0.66	1			7.17E+12
		20	400.81	124.12	46	830	127.33	412	7440	127.33	0.95	1			7.75E+13
		24	400.81	125.05	53	948	126.86	454	7776	126.86	2.33	3			2.3E+14
	51-75-75	4	375.01	90.59	15	294	90.59	79	1420	91.99	0.81	2			1215450
		8	375.01	144.53	31	572	145.90	216	3752	149.87	3.41	6			1.69E+10
		12	375.01	178.17	50	834	188.43	586	10584	188.43	4.99	9			2.61E+13
		16	375.01	190.51	63	1080	200.16	759	13216	202.47	3.75	7			8.55E+15

Table D-4. Computational Results for the TSP instance, eil51. (cont'd)

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil51	51-25-25	4*	46.20	0.00	0.00	0.00	5	22	0.45	3285	0.74	3.32
		6*	58.89	0.00	4.97	0.00	7	31	0.44	54533	0.08	0.32
		8	66.31	0.00	0.00	0.00	8	80	0.73		0.02	0.13
		10	70.13	0.00	1.25	0.00	10	46	0.44		0.01	0.02
		12	69.87	0.06	0.44	0.00	12	66	2.23		0.00	0.02
	51-51-51	4*	53.42	0.00	0.00	0.00	10	40	0.45	71859.00	0.08	0.30
		8	86.26	0.00	0.00	0.00	20	179	0.80		0.00	0.00
		12	107.14	0.47	0.78	0.00	30	301	0.98		0.00	0.00
		16	120.08	2.60	0.00	0.00	38	238	0.66		0.00	0.00
		20	127.33	2.53	0.00	0.00	46	412	0.95		0.00	0.00
		24	126.86	1.42	0.00	0.00	53	454	2.33		0.00	0.00
	51-75-75	4	91.99	1.53	1.53	0.00	15	79	0.81		0.02	0.12
		8	149.87	3.57	2.65	0.00	31	216	3.41		0.00	0.00
		12	188.43	5.45	0.00	0.00	50	586	4.99		0.00	0.00
		16	202.47	5.91	1.14	0.00	63	759	3.75		0.00	0.00
* f(best) is the optimum.			Average	1.57	0.85	0.00	27	234	1.56	43226	0.06	0.28
			Maximum	5.91	4.97	0.00	63	759	4.99	71859	0.74	3.32
			# Best	6	8	15						
			Total	15								

Table D-5. Computational Results for the TSP instance, eil101.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
eil101	101-20-20	4	692.57	39.12	6.7	74	39.12	17	192	39.12	0.44	1	39.12	1340	4845
		6	692.57	47.27	9.5	105	47.27	31	336	47.27	0.75	1	47.27	11685	38760
		8	692.57	49.62	12.0	132	49.62	75	768	49.62	0.81	1	49.62	42879	125970
		10	692.57	48.06	14.1	155	48.06	38	400	48.06	0.69	1	48.06	74922	184756
	101-40-40	4	717.97	49.55	13.5	154	49.55	52	576	49.97	1.22	2	49.97	39287	91390
		8	717.97	75.20	25.6	292	75.20	113	1280	76.45	0.95	1			76904685
		12	717.97	87.02	36.1	414	87.06	335	3696	87.06	0.99	1			5.59E+09
		16	717.97	94.87	45.8	520	94.92	372	3840	94.92	2.92	5			6.29E+10
		20	717.97	96.23	55.4	610	96.23	349	3600	96.23	1.33	1			1.38E+11
	101-60-60	4	771.67	71.96	21.0	234	71.96	103	1120	71.96	0.45	1			487635
		8	771.67	113.93	40.7	452	114.73	441	4576	114.73	0.84	1			2.56E+09
		12	771.67	137.85	59.7	654	139.18	604	6336	139.18	0.83	1			1.4E+12
		16	771.67	147.53	77.5	840	149.76	981	9856	149.76	11.62	20			1.5E+14
		20	771.67	149.83	93.8	1010	148.09	1298	12800	151.01	3.48	2			4.19E+15
	101-80-80	4	821.77	110.27	27.1	314	110.27	164	1824	110.27	0.55	1			1581580
		8	821.77	165.90	53.5	612	165.90	467	5184	168.92	5.72	8			2.9E+10
		12	821.77	196.70	79.2	894	198.64	998	10608	200.85	5.25	6			6.02E+13
		16	821.77	216.76	103.1	1160	218.21	1571	16384	219.80	10.56	15			2.7E+16

Table D-5. Computational Results for the TSP instance, eil101. (cont'd)

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil101	101-20-20	4*	39.12	0.00	0.00	0.00	6.7	17	0.44	1340	1.53	3.96
		6*	47.27	0.00	0.00	0.00	9.5	31	0.75	11685	0.27	0.87
		8*	49.62	0.00	0.00	0.00	12.0	75	0.81	42879	0.10	0.61
		10*	48.06	0.00	0.00	0.00	14.1	38	0.69	74922	0.08	0.22
	101-40-40	4*	49.97	0.83	0.83	0.00	13.5	52	1.22	39287	0.17	0.63
		8	76.45	1.63	1.63	0.00	25.6	113	0.95		0.00	0.00
		12	87.06	0.05	0.00	0.00	36.1	335	0.99		0.00	0.00
		16	94.92	0.05	0.00	0.00	45.8	372	2.92		0.00	0.00
		20	96.23	0.00	0.00	0.00	55.4	349	1.33		0.00	0.00
	101-60-60	4	71.96	0.00	0.00	0.00	21.0	103	0.45		0.05	0.23
		8	114.73	0.70	0.00	0.00	40.7	441	0.84		0.00	0.00
		12	139.18	0.96	0.00	0.00	59.7	604	0.83		0.00	0.00
		16	149.76	1.49	0.00	0.00	77.5	981	11.62		0.00	0.00
		20	151.01	0.78	1.94	0.00	93.8	1298	3.48		0.00	0.00
	101-80-80	4	110.27	0.00	0.00	0.00	27.1	164	0.55		0.02	0.12
		8	168.92	1.79	1.79	0.00	53.5	467	5.72		0.00	0.00
		12	200.85	2.07	1.10	0.00	79.2	998	5.25		0.00	0.00
		16	219.80	1.38	0.72	0.00	103.1	1571	10.56		0.00	0.00
* f(best) is the optimum.			Average	0.65	0.44	0.00	43	445	2.74	34023	0.12	0.37
			Maximum	2.07	1.94	0.00	103	1571	11.62	74922	1.53	3.96
			# Best	7	12	18						
			Total	18								

APPENDIX E

COMPUTATIONAL RESULTS FOR VRP INSTANCES

In this appendix, computational results for all VRP instances that comprise eil22, eil23, eil30, eil33, att48, eil76 and eil101 are given in detail. All of the instances are composed of 3 different problems and each problem has been solved for different number of bottle banks with the maximum number of 25 for eil76 and eil101. CPU time, percent deviation from the best solution and percentage of all possible TSPs solved are presented for each *p*. The results are given for VNS, Local Search and Greedy Heuristic algorithms and where applicable, solution results for Exhaustive Search are also provided.

Table E-1. Computational Results for the VRP instance, eil22.

				GH+C			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
eil22	4 54 18	4*	4.78	99.56	2	66	102.75	8	224	102.75	0.22	1	102.75	100	3060
		6	4.78	110.22	3	93	103.80	5	144	113.46	0.31	1	113.46	967	18564
		8	4.78	114.22	4	116	114.43	17	480	114.43	0.27	1	114.43	2518	43758
	2 60 20	4*	-15.36	74.82	2	74	74.81	9	256	80.06	0.22	1	80.06	164	4845
		6*	-15.36	92.24	4	105	98.16	8	252	101.22	0.19	1	101.22	1352	38760
		8	-15.36	104.37	5	132	111.01	13	384	111.01	0.30	1	111.01	8871	125970
		10	-15.36	110.35	6	155	111.01	18	500	111.01	0.22	1	111.01	10023	184756
	1 63 21	4*	11.50	64.81	3	78	64.81	10	340	64.81	0.28	1	64.81	198	5985
		6*	11.50	69.66	4	111	70.34	12	360	70.34	0.28	1	70.34	1846	54264
		8	11.50	74.43	5	140	74.99	21	624	74.99	0.31	1	74.99	6901	203490
		10	11.50	74.06	6	165	77.48	34	990	77.48	0.28	1	77.48	19014	352716

* In the Local Search, Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil22	4 54 18	4*	102.75	3.11	0.00	0.00	2	8	0.22	100	2.16	7.32
		6*	113.46	2.86	8.52	0.00	3	5	0.31	967	0.50	0.78
		8*	114.43	0.18	0.00	0.00	4	17	0.27	2518	0.27	1.10
	2 60 20	4*	80.06	6.55	6.57	0.00	2	9	0.22	164	1.53	5.28
		6*	101.22	8.88	3.02	0.00	4	8	0.19	1352	0.27	0.65
		8*	111.01	5.98	0.00	0.00	5	13	0.30	8871	0.10	0.30
		10*	111.01	0.60	0.00	0.00	6	18	0.22	10023	0.08	0.27
	1 63 21	4*	64.81	0.00	0.00	0.00	3	10	0.28	198	1.30	5.68
		6*	70.34	0.97	0.00	0.00	4	12	0.28	1846	0.20	0.66
		8*	74.99	0.75	0.00	0.00	5	21	0.31	6901	0.07	0.31
		10*	77.48	4.41	0.00	0.00	6	34	0.28	19014	0.05	0.28
* f(best) is the optimum.			Average	3.12	1.65	0.00	4	14	0.26	4723	0.59	2.06
			Maximum	8.88	8.52	0.00	6	34	0.31	19014	2.16	7.32
			# Best	1	8	11						
			Total	11								

Table E-2. Computational Results for the VRP instance, eil23.

				GH+C			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
eil23	5 54 18	4	-26.90	73.37	2	66	73.37	10	280	73.37	0.16	1	73.37	103	3060
		6	-26.90	85.17	4	93	85.17	10	288	85.27	0.22	1	85.27	944	18564
		8	-26.90	89.39	5	116	89.42	16	480	89.42	0.31	1	89.42	2654	43758
	2 63 21	4*	-17.01	72.32	3	78	74.91	13	408	74.91	0.16	1	74.91	220	5985
		6*	-17.01	79.77	4	111	79.77	15	450	79.77	0.19	1	79.77	2258	54264
		8	-17.01	74.88	5	140	74.88	25	728	74.88	0.28	1	74.88	10171	203490
		10	-17.01	68.00	7	165	68.00	27	770	68.00	0.53	2	68.00	18072	352716
	1 66 22	4*	15.60	20.48	3	82	29.58	12	360	29.58	0.16	1	29.58	244	7315
		6*	15.60	16.20	4	117	16.20	13	384	42.97	0.22	1	42.97	2664	74613
		8	15.60	10.30	5	148	43.21	26	784	43.21	0.28	1	43.21	10875	319770
		10	15.60	26.83	6	175	40.14	26	720	40.14	0.22	1	40.14	32422	646646

* In the Local Search, Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil23	5 54 18	4*	73.37	0.00	0.00	0.00	2	10	0.16	103	2.16	9.15
		6*	85.27	0.12	0.12	0.00	4	10	0.22	944	0.50	1.55
		8*	89.42	0.03	0.00	0.00	5	16	0.31	2654	0.27	1.10
	2 63 21	4*	74.91	3.46	0.00	0.00	3	13	0.16	220	1.30	6.82
		6*	79.77	0.00	0.00	0.00	4	15	0.19	2258	0.20	0.83
		8*	74.88	0.00	0.00	0.00	5	25	0.28	10171	0.07	0.36
		10*	68.00	0.00	0.00	0.00	7	27	0.53	18072	0.05	0.22
	1 66 22	4*	29.58	30.77	0.00	0.00	3	12	0.16	244	1.12	4.92
		6*	42.97	62.31	62.31	0.00	4	13	0.22	2664	0.16	0.51
		8*	43.21	76.16	0.00	0.00	5	26	0.28	10875	0.05	0.25
		10*	40.14	33.17	0.00	0.00	6	26	0.22	32422	0.03	0.11
* f(best) is the optimum.			Average	18.73	5.67	0.00	4	18	0.25	7330	0.54	2.35
			Maximum	76.16	62.31	0.00	7	27	0.53	32422	2.16	9.15
			# Best	4	9	11						
			Total	11								

Table E-3. Computational Results for the VRP instance, eil30.

				GH+C			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
eil30	6 72 24	4	13.58	53.10	3	90	53.10	8	240	53.10	0.17	1	53.10	649	10626
		8	13.58	54.72	7	164	54.72	19	512	54.72	0.48	1	54.72	53754	735471
		12	13.58	53.39	10	222	53.85	45	1008	53.85	0.49	1			2704156
	3 81 27	4*	-23.97	93.51	3	102	93.51	21	644	93.51	0.20	1	93.51	752	17550
		8	-23.97	112.46	7	188	115.59	37	1064	115.59	0.34	1			2220075
		12	-23.97	111.12	10	258	114.48	68	1800	114.48	0.53	1			17383860
	1 87 29	4*	10.10	27.83	4	110	21.67	16	500	27.83	0.17	1	27.83	1084	23751
		8	10.10	29.01	7	204	36.37	46	1344	36.37	0.41	1			4292145
		12	10.10	27.33	11	282	35.12	52	1428	35.12	0.61	1			51895935
		14	10.10	31.77	13	315	31.77	65	1680	34.28	0.34	1			77558760

* In the Local Search, Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil30	6 72 24	4*	53.10	0.00	0.00	0.00	3	8	0.17	649	0.85	2.26
		8*	54.72	0.00	0.00	0.00	7	19	0.48	53754	0.02	0.07
		12	53.85	0.85	0.00	0.00	10	45	0.49		0.01	0.04
	3 81 27	4*	93.51	0.00	0.00	0.00	3	21	0.20	752	0.58	3.67
		8	115.59	2.71	0.00	0.00	7	37	0.34		0.01	0.05
		12	114.48	2.94	0.00	0.00	10	68	0.53		0.00	0.01
	1 87 29	4*	27.83	0.00	22.11	0.00	4	16	0.17	1084	0.46	2.11
		8	36.37	20.26	0.00	0.00	7	46	0.41		0.00	0.03
		12	35.12	22.16	0.00	0.00	11	52	0.61		0.00	0.00
		14	34.28	7.33	7.33	0.00	13	65	0.34		0.00	0.00
* f(best) is the optimum.			Average	5.62	2.94	0.00	8	38	0.38	14060	0.19	0.82
			Maximum	22.16	22.11	0.00	13	68	0.61	53754	0.85	3.67
			# Best	4	8	10						
			Total	10								

Table E-4. Computational Results for the VRP instance, eil33.

				GH+C			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
eil33	7 78 26	4	4.88	83.29	4	98	83.29	13	352	83.29	0.24	1	83.29	814	14950
		8	4.88	107.92	8	180	108.28	37	1008	108.28	0.52	1			1562275
		12	4.88	107.65	12	246	117.28	67	1680	117.28	0.45	1			9657700
	3 90 30	4*	-24.18	81.50	4	114	81.50	14	416	81.50	0.16	1	81.50	1949	27405
		8	-24.18	101.51	7	212	101.51	56	1584	101.51	0.33	1			5852925
		12	-24.18	102.74	11	294	102.46	70	1944	102.74	1.45	5			86493225
		14	-24.18	100.46	13	329	82.84	50	1344	101.98	0.47	1			1.45E+08
	1 96 32	4*	13.40	14.88	4	122	16.33	18	560	16.52	0.19	1	16.52	3496	35960
		8	13.40	29.91	7	228	35.89	58	1728	35.89	0.30	1			10518300
		12	13.40	28.55	12	318	33.21	74	2160	35.28	0.50	1			2.26E+08
		16	13.40	28.25	16	392	33.68	68	1792	33.68	0.89	2			6.01E+08

* In the Local Search, Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
eil33	7 78 26	4*	83.29	0.00	0.00	0.00	4	13	0.24	814	0.66	2.35
		8	108.28	0.34	0.00	0.00	8	37	0.52		0.01	0.06
		12	117.28	8.21	0.00	0.00	12	67	0.45		0.00	0.02
	3 90 30	4*	81.50	0.00	0.00	0.00	4	14	0.16	1949	0.42	1.52
		8	101.51	0.00	0.00	0.00	7	56	0.33		0.00	0.03
		12	102.74	0.00	0.27	0.00	11	70	1.45		0.00	0.00
		14	101.98	1.50	18.77	0.00	13	50	0.47		0.00	0.00
	1 96 32	4*	16.52	9.91	1.17	0.00	4	18	0.19	3496	0.34	1.56
		8	35.89	16.65	0.00	0.00	7	58	0.30		0.00	0.02
		12	35.28	19.06	5.85	0.00	12	74	0.50		0.00	0.00
		16	33.68	16.14	0.00	0.00	16	68	0.89		0.00	0.00
* f(best) is the optimum.			Average	6.53	2.37	0.00	9	48	0.50	2086	0.13	0.51
			Maximum	19.06	18.77	0.00	16	74	1.45	3496	0.66	2.35
			# Best	4	7	11						
			Total	11								

Table E-5. Computational Results for the VRP instance, att48.

				GH+C			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
att48	10 114 38	5*	-24.03	165.24	7	180	165.24	36	990	165.24	0.19	1	165.24	28821	501942
		10*	-24.03	218.24	13	335	222.43	66	1680	225.62	2.17	8			4.73E+08
		15*	-24.03	224.71	17	465	228.96	142	3795	229.93	1.22	2			1.55E+10
	5 129 43	5	-62.26	118.40	8	205	121.68	40	1140	121.68	0.19	1	121.68	50596	962598
		10	-62.26	167.14	17	385	175.79	81	2310	178.57	0.31	1			1.92E+09
		15	-62.26	178.03	24	540	186.45	167	4620	186.45	3.25	6			1.52E+11
		20	-62.26	169.92	33	670	178.63	119	3220	178.63	0.73	1			9.61E+11
	1 141 47	5**	17	60.40	8	225	75.90	35	1050	75.90	0.22	1	75.90	76593	1533939
		10	17	84.07	16	425	93.89	130	3700	104.64	0.98	3			5.18E+09
		15	17	85.19	24	600	118.02	213	5760	120.38	0.83	1			7.52E+11
		20	17	108.23	33	750	114.29	276	7560	114.29	2.42	2			9.76E+12

* In the Greedy Heuristic, Linkern has been used instead of Concorde.

** In the Local Search, Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
att48	10 114 38	5*	165.24	0.00	0.00	0.00	7	36	0.19	28821	0.04	0.20
		10	225.62	3.27	1.41	0.00	13	66	2.17		0.00	0.00
		15	229.93	2.27	0.42	0.00	17	142	1.22		0.00	0.00
	5 129 43	5*	121.68	2.69	0.00	0.00	8	40	0.19	50596	0.02	0.12
		10	178.57	6.40	1.56	0.00	17	81	0.31		0.00	0.00
		15	186.45	4.52	0.00	0.00	24	167	3.25		0.00	0.00
		20	178.63	4.88	0.00	0.00	33	119	0.73		0.00	0.00
	1 141 47	5*	75.90	20.42	0.00	0.00	8	35	0.22	76593	0.01	0.07
		10	104.64	19.66	10.28	0.00	16	130	0.98		0.00	0.00
		15	120.38	29.23	1.96	0.00	24	213	0.83		0.00	0.00
		20	114.29	5.30	0.00	0.00	33	276	2.42		0.00	0.00
* f(best) is the optimum.			Average	8.97	1.42	0.00	18	119	1.14	52003	0.01	0.03
			Maximum	29.23	10.28	0.00	33	276	3.25	76593	0.04	0.20
			# Best	1	6	11						
			Total	11								

Table E-6. Computational Results for the VRP instance, eil76.

				GH+L			LS+L			VNS+L+I			
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Total # of TSPs
eil76	15 183 61	10	36.10	253.09	21	565	254.39	132	3570	254.39	0.42	1	9E+10
		15	36.10	308.95	30	810	304.92	261	6900	314.72	0.98	1	7.1E+13
		20	36.10	335.35	39	1030	337.21	566	13940	341.66	12.16	7	6.2E+15
		25	36.10	343.59	47	1225	352.17	619	14400	352.17	3.29	1	8.8E+16
	8 204 68	10	-19.75	280.65	22	635	280.65	247	6960	280.65	0.59	1	2.9E+11
		15	-19.75	340.54	33	915	340.52	475	12720	340.54	1.61	1	4.4E+14
		20	-19.75	365.97	42	1170	361.72	642	16320	367.74	2.64	1	8.2E+16
		25	-19.75	370.65	51	1400	366.86	713	18275	371.02	27.53	7	2.6E+18
	1 225 75	10*	17.30	165.74	26	705	177.49	269	7800	185.66	0.75	1	8.3E+11
		15*	17.30	238.40	41	1020	245.55	437	12600	245.55	9.66	8	2.3E+15
		20*	17.30	272.06	57	1310	277.02	401	11000	285.59	2.02	1	8E+17
		25*	17.30	295.15	74	1575	299.23	652	17500	303.27	20.20	6	5.3E+19

* In the Greedy Heuristic Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)			% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	GH	LS
eil76	15 183 61	10	254.39	0.51	0.00	0.00	21	132	0.42	0.00	0.00
		15	314.72	1.84	3.11	0.00	30	261	0.98	0.00	0.00
		20	341.66	1.85	1.30	0.00	39	566	12.16	0.00	0.00
		25	352.17	2.44	0.00	0.00	47	619	3.29	0.00	0.00
	8 204 68	10	280.65	0.00	0.00	0.00	22	247	0.59	0.00	0.00
		15	340.54	0.00	0.00	0.00	33	475	1.61	0.00	0.00
		20	367.74	0.48	1.64	0.00	42	642	2.64	0.00	0.00
		25	371.02	0.10	1.12	0.00	51	713	27.53	0.00	0.00
	1 225 75	10	185.66	10.73	4.40	0.00	26	269	0.75	0.00	0.00
		15	245.55	2.91	0.00	0.00	41	437	9.66	0.00	0.00
		20	285.59	4.74	3.00	0.00	57	401	2.02	0.00	0.00
		25	303.27	2.68	1.33	0.00	74	652	20.20	0.00	0.00
			Average	2.36	1.33	0.00	40	451	6.82	0.00	0.00
			Maximum	10.73	4.40	0.00	74	713	27.53	0.00	0.00
			# Best	2	4	12					
			Total	12							

Table E-7. Computational Results for the VRP instance, eil101.

				GH+L			LS+L			VNS+L+I			
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Total # of TSPs
eil101	20 243 81	10	57.36	430.53	28	765	406.46	353	9230	430.53	2.13	3	1.88E+12
		15	57.36	509.72	42	1110	501.21	443	10890	519.17	2.33	1	8.14E+15
		20	57.36	548.11	54	1430	552.67	574	13420	565.00	5.18	2	4.69E+18
		25	57.36	562.87	67	1725	583.40	1265	29400	585.43	39.37	11	5.26E+20
	10 273 91	10	7.96	441.94	31	865	449.67	310	8910	449.67	1.34	1	6.43E+12
		15	7.96	547.35	45	1260	544.23	411	11400	566.97	3.59	1	5.48E+16
		20	7.96	605.52	59	1630	602.84	1097	28400	615.50	4.81	1	6.53E+19
		25	7.96	621.23	72	1975	628.36	1490	37950	632.78	9.88	1	1.6E+22
	1 300 100	10*	14.90	355.43	34	955	360.26	274	8100	369.62	1.39	1	1.92E+13
		15*	14.90	446.15	58	1395	473.16	786	22950	473.16	28.24	10	2.98E+17
		20*	14.90	516.63	78	1810	535.29	1136	32000	535.29	13.67	3	6.68E+20
		25*	14.90	546.53	103	2200	557.32	1082	30000	561.74	80.28	13	3.22E+23

* In the Greedy Heuristic Concorde has been used instead of Linkern.

				% Deviation from Best			CPU (seconds)			% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	GH	LS
eil101	20 243 81	10	430.53	0.00	5.59	0.00	28	353	2.13	0.00	0.00
		15	519.17	1.82	3.46	0.00	42	443	2.33	0.00	0.00
		20	565.00	2.99	2.18	0.00	54	574	5.18	0.00	0.00
		25	585.43	3.85	0.35	0.00	67	1265	39.37	0.00	0.00
	10 273 91	10	449.67	1.72	0.00	0.00	31	310	1.34	0.00	0.00
		15	566.97	3.46	4.01	0.00	45	411	3.59	0.00	0.00
		20	615.50	1.62	2.06	0.00	59	1097	4.81	0.00	0.00
		25	632.78	1.82	0.70	0.00	72	1490	9.88	0.00	0.00
	1 300 100	10	369.62	3.84	2.53	0.00	34	274	1.39	0.00	0.00
		15	473.16	5.71	0.00	0.00	58	786	28.24	0.00	0.00
		20	535.29	3.49	0.00	0.00	78	1136	13.67	0.00	0.00
		25	561.74	2.71	0.79	0.00	103	1082	80.28	0.00	0.00
			Average	2.75	1.81	0.00	56	768	16.02	0.00	0.00
			Maximum	5.71	5.59	0.00	103	1490	80.28	0.00	0.00
			# Best	1	3	12					
			Total	12							

APPENDIX F

COMPUTATIONAL RESULTS FOR RANDOM INSTANCES

In this appendix, computational results for all random instances that comprise ran20, ran30, ran40, ran50, ran60, ran80 and ran100 are given in detail. There is one problem for each instance and each problem has been solved for four different number of bottle banks. CPU time, percent deviation from the best solution and percentage of all possible TSPs solved are presented for each *p*. The results are given for VNS, Local Search and Greedy Heuristic algorithms and where applicable, solution results for Exhaustive Search are also provided.

Table F-1. Computational Results for Random instances, ran20, ran30, ran40, ran50.

				GH+L			LS+L			VNS+L+I			ES+C		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step	Optimal Profit(P)	CPU (s)	Total # of TSPs
ran20	10 60 20	4	-24.85	75.39	3	74	74.20	8	256	75.39	0.39	2	75.39	242	4845
		6	-24.85	101.08	4	105	101.08	17	504	101.08	0.28	1	101.08	2080	38760
		8	-24.85	113.54	5	132	113.82	17	480	113.82	1.02	2	113.82	7187	125970
		10	-24.85	118.75	6	155	118.04	18	500	119.05	0.27	1	119.05	10984	184756
ran30	15 90 30	4	41.55	126.75	4	114	126.75	15	416	126.75	0.25	1	126.75	1732	27405
		8	41.55	194.11	8	212	194.11	46	1232	194.11	0.38	1			5852925
		12	41.55	208.15	11	294	209.80	74	1944	210.73	0.38	1			86493225
		15	41.55	207.38	13	345	208.73	80	2025	208.73	0.42	1			1.55E+08
ran40	20 120 40	5	118.25	137.02	7	190	137.02	39	1050	137.02	0.20	1	137.02	38348	658008
		10	118.25	199.86	13	355	201.44	105	2700	201.44	0.44	1			8.48E+08
		15	118.25	216.56	18	495	218.35	137	3375	218.35	0.88	1			4.02E+10
		20	118.25	217.29	23	610	217.19	192	4400	217.29	14.97	33			1.38E+11
ran50	25 150 50	5	141.35	187.93	10	240	187.93	59	1350	187.93	0.48	1	187.93	206371	2118760
		10	141.35	300.67	20	455	300.19	131	2800	300.67	2.55	7			1.03E+10
		15	141.35	357.13	32	645	356.18	278	5775	361.28	1.03	1			2.25E+12
		20	141.35	383.98	36	810	386.05	360	6600	387.73	3.23	3			4.71E+13

Table F-1. Computational Results for Random instances, ran20, ran30, ran40, ran50. (cont'd)

				% Deviation from Best			CPU (seconds)				% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	ES	GH	LS
ran20	10 60 20	4*	75.39	0.00	1.57	0.00	3	8	0.39	242	1.53	5.28
		6*	101.08	0.00	0.00	0.00	4	17	0.28	2080	0.27	1.30
		8*	113.82	0.24	0.00	0.00	5	17	1.02	7187	0.10	0.38
		10*	119.05	0.25	0.85	0.00	6	18	0.27	10984	0.08	0.27
ran30	15 90 30	4*	126.75	0.00	0.00	0.00	4	15	0.25	1732	0.42	1.52
		8	194.11	0.00	0.00	0.00	8	46	0.38		0.00	0.02
		12	210.73	1.22	0.44	0.00	11	74	0.38		0.00	0.00
		15	208.73	0.65	0.00	0.00	13	80	0.42		0.00	0.00
ran40	20 120 40	5*	137.02	0.00	0.00	0.00	7	39	0.20	38348	0.03	0.16
		10	201.44	0.79	0.00	0.00	13	105	0.44		0.00	0.00
		15	218.35	0.82	0.00	0.00	18	137	0.88		0.00	0.00
		20	217.29	0.00	0.05	0.00	23	192	14.97		0.00	0.00
ran50	25 150 50	5*	187.93	0.00	0.00	0.00	10	59	0.48	206371	0.01	0.06
		10	300.67	0.00	0.16	0.00	20	131	2.55		0.00	0.00
		15	361.28	1.15	1.41	0.00	32	278	1.03		0.00	0.00
		20	387.73	0.97	0.43	0.00	36	360	3.23		0.00	0.00
* f(best) is the optimum.			Average	0.38	0.31	0.00	13	99	1.70	38135	0.15	0.56
			Maximum	1.22	1.57	0.00	36	360	14.97	206371	1.53	5.28
			# Best	8	9	16						
			Total	16								

Table F-2. Computational Results for Random instances, ran60, ran80, ran100.

				LS+L			GH+L			VNS+L+I		
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# of TSP solved	Profit(P)	CPU (s)	# main step
ran60	30 180 60	5	-12.80	200.20	11	290	200.20	54	1375	207.67	0.266	1
		10	-12.80	333.48	22	555	344.64	183	4500	344.64	1.641	3
		15	-12.80	402.61	32	795	415.85	393	9450	415.85	1.046	1
		20	-12.80	441.80	41	1010	444.11	496	12000	444.11	1.515	1
ran80	40 240 80	5	92.30	261.86	17	390	261.86	107	2250	261.86	0.422	1
		10	92.30	440.97	35	755	440.97	527	11200	440.97	0.75	1
		15	92.30	541.59	49	1095	541.59	483	10725	541.59	2	1
		20	92.30	580.53	61	1410	583.12	1020	20400	583.12	21.53	10
ran100	50 300 100	5	180.70	309.47	24	490	309.47	155	2850	309.47	1	2
		10	180.70	518.07	48	955	523.83	561	9000	528.12	3.094	1
		15	180.70	665.02	73	1395	684.53	1092	17850	684.53	14.407	4
		20	180.70	734.02	97	1810	761.91	2030	32000	773.51	113.441	35

Table F2. Computational Results for Random instances, ran60, ran80, ran100. (cont'd)

				% Deviation from Best			CPU (seconds)			% TSP Solved	
Problem set	*m-n-s*	*p*	f(best)	GH	LS	VNS	GH	LS	VNS	GH	LS
ran60	30 180 60	5	207.67	3.60	3.60	0.00	11	54	0.27	0.01	0.03
		10	344.64	3.24	0.00	0.00	22	183	1.64	0.00	0.00
		15	415.85	3.18	0.00	0.00	32	393	1.05	0.00	0.00
		20	444.11	0.52	0.00	0.00	41	496	1.52	0.00	0.00
ran80	40 240 80	5	261.86	0.00	0.00	0.00	17	107	0.42	0.00	0.01
		10	440.97	0.00	0.00	0.00	35	527	0.75	0.00	0.00
		15	541.59	0.00	0.00	0.00	49	483	2.00	0.00	0.00
		20	583.12	0.44	0.00	0.00	61	1020	21.53	0.00	0.00
ran100	50 300 100	5	309.47	0.00	0.00	0.00	24	155	1.00	0.00	0.00
		10	528.12	1.90	0.81	0.00	48	561	3.09	0.00	0.00
		15	684.53	2.85	0.00	0.00	73	1092	14.41	0.00	0.00
		20	773.51	5.11	1.50	0.00	97	2030	113.44	0.00	0.00
			Average	1.74	0.49	0.00	42	592	13.43	0.00	0.00
			Maximum	5.11	3.60	0.00	97	2030	113.44	0.01	0.03
			# Best	4	9	12					
			Total	12							

APPENDIX G

COMPUTATIONAL RESULTS FOR LARGE INSTANCES

In this appendix, computational results for the large instances, gil262 and KroA200 are given in detail. While gil262 instance set is composed of 3 problems, KroA200 set is composed of 5. The maximum number of bottle banks to locate is 30. CPU time and percent deviation from the best solution are presented for each *p*. The results are given for VNS, RVNS, Local Search and Greedy Heuristic algorithms. The solution results do not include Exhaustive Search algorithm since the solution space is very large and it is inappropriate to utilize the algorithm.

Table G-1. Computational Results for the large instance gil262.

				GH+L		LS+L		RVNS+L (GH)		RVNS+L (LS)		VNS+L+I	
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	# of TSP solved	Profit(P)	# of TSP solved	Profit(P)	# of main step	Profit(P)	# of main step	Profit(P)	# main step
gil262	52 630 210	10	-26.61	799.95	2055	814.97	24000	798.65	202	816.44	2458	818.70	3
		20	-26.61	1340.26	4010	1375.60	98800	1213.07	170	1357.62	4435	1380.25	1
		30	-26.61	1575.55	5865	1610.66	178200	1430.84	153	1557.51	4677	1610.66	7
	26 708 236	10	-217.82	941.60	2315	947.12	29380	841.93	206	947.12	3044	947.12	11
		20	-217.82	1420.88	4530	1452.08	82080	1336.77	176	1420.23	3621	1468.48	8
		30	-217.82	1671.00	6645	1718.32	173040	1599.90	165	1687.09	4817	1739.53	32
	1 783 261	10*	17.2	656.75	2565	662.61	32630	618.83	356	672.13	4400	679.29	21
		20*	17.2	1158.26	5030	1202.34	120500	1061.07	311	1176.90	8695	1207.74	7
		30*	17.2	1426.80	7395	1463.92	194040	1362.86	279	1451.40	9535	1486.51	22

* For these instances Concorde.exe has been utilized in Greedy Heuristic Algorithm instead of Linkern.exe.

				% Deviation from Best					CPU (seconds)				
Problem set	*m-n-s*	*p*	f(best)	GH	LS	RVNS-GH	RVNS-LS	VNS	GH	LS	RVNS-GH	RVNS-LS	VNS
gil262	52 630 210	10	818.70	2.29	0.46	2.45	0.28	0.00	125	1540	125	1540	10.81
		20	1380.25	2.90	0.34	12.11	1.64	0.00	236	6569	237	6569	28.95
		30	1610.66	2.18	0.00	11.16	3.30	0.00	351	12051	353	12054	499.94
	26 708 236	10	947.12	0.58	0.00	11.11	0.00	0.00	98	1389	98	1389	38.53
		20	1468.48	3.24	1.12	8.97	3.29	0.00	198	4091	199	4092	191.20
		30	1739.53	3.94	1.22	8.03	3.01	0.00	309	8994	310	8994	2470.58
	1 783 261	10*	679.29	3.32	2.45	8.90	1.05	0.00	91	1138	125	1540	85.17
		20*	1207.74	4.10	0.45	12.14	2.55	0.00	208	4381	238	6569	253.19
		30*	1486.51	4.02	1.52	8.32	2.36	0.00	342	7417	351	12053	2334.69
			Average	2.95	0.84	9.24	1.94	0.00	218	5286	226	6089	657.01
			Maximum	4.10	2.45	12.14	3.30	0.00	351	12051	353	12054	2470.58
			# Best	0	2	0	1	9					
			Total	9									

Table G-2. Computational Results for the large instance KroA200.

				GH+L		LS+L		RVNS+L (GH)		RVNS+L (LS)		VNS+L+I	
Problem set	*m-n-s*	*p*	Profit(0)	Profit(P)	# of TSP solved	Profit(P)	# of TSP solved	Profit(P)	# of main step	Profit(P)	# of main step	Profit(P)	# main step
kroA200	200 30 30	4	1513.10	44.73	114	44.89	520	42.97	25	43.75	124	44.89	1
		6	1513.10	53.70	165	53.86	720	48.36	23	49.66	112	53.86	2
		8	1513.10	59.52	212	60.29	1056	56.78	23	57.66	126	60.29	1
		10	1513.10	63.42	255	64.19	1800	58.68	23	64.19	172	64.19	1
		12	1513.10	62.50	294	63.27	2160	57.00	22	62.55	174	63.27	1
		14	1513.10	60.89	329	61.66	1568	56.28	20	60.23	108	61.66	1
	200 60 60	4	1540.50	60.43	234	60.43	1120	54.04	58	60.43	274	60.43	1
		8	1540.50	93.67	452	95.08	4160	89.04	57	95.08	538	95.10	10
		12	1540.50	117.90	654	123.26	6912	109.05	53	119.41	618	123.26	1
		16	1540.50	127.52	840	133.35	7744	116.93	51	128.36	473	133.35	9
		20	1540.50	128.43	1010	133.71	13600	126.54	45	132.71	658	133.71	1
	200 90 90	4	1436.40	109.51	354	108.93	1032	109.47	74	109.51	275	109.51	1
		8	1436.40	186.18	692	186.18	4592	162.68	76	182.76	623	187.28	1
		12	1436.40	229.44	1014	226.06	10296	202.32	64	228.24	851	229.55	15
		16	1436.40	244.16	1320	245.82	22496	227.77	62	244.45	1449	246.32	6
	200 120 120	4	1503.30	148.86	474	148.86	2320	141.88	110	144.40	565	148.86	1
		8	1503.30	251.04	932	255.17	8064	226.47	116	245.34	988	255.17	1
		12	1503.30	305.01	1374	306.28	15552	273.93	119	303.80	1362	306.45	9
		16	1503.30	328.02	1800	331.97	36608	297.31	101	328.36	2106	332.52	13
	200 150 150	4	1474.50	179.83	594	179.83	2920	167.85	128	177.95	663	179.83	34
		8	1474.50	303.72	1172	306.10	11360	278.58	145	302.07	1424	306.10	8
		12	1474.50	370.74	1734	381.06	19872	341.51	153	366.29	1699	381.06	7
		16	1474.50	412.63	2280	416.87	49312	371.81	128	403.05	2797	421.40	6

Table G-2. Computational Results for the large instance KroA200. (cont'd)

				% Deviation from Best					CPU (seconds)				
Problem set	*m-n-s*	*p*	f(best)	GH	LS	RVNS-GH	RVNS-LS	VNS	GH	LS	RVNS-GH	RVNS-LS	VNS
kroA200	200 30 30	4	44.89	0.34	0.00	4.28	2.53	0.00	23	106	24	106	1.31
		6	53.86	0.29	0.00	10.21	7.79	0.00	34	148	34	148	3.69
		8	60.29	1.27	0.00	5.82	4.37	0.00	43	222	45	224	2.02
		10	64.19	1.20	0.00	8.59	0.00	0.00	53	380	54	381	1.64
		12	63.27	1.21	0.00	9.90	1.13	0.00	61	470	63	472	2.81
		14	61.66	1.25	0.00	8.72	2.33	0.00	69	349	69	351	2.09
	200 60 60	4	60.43	0.00	0.00	10.56	0.00	0.00	47	219	47	220	1.88
		8	95.10	1.50	0.02	6.38	0.02	0.00	93	874	95	875	15.06
		12	123.26	4.35	0.00	11.52	3.12	0.00	135	1501	137	1502	1.86
		16	133.35	4.37	0.00	12.31	3.74	0.00	176	1588	179	1588	12.41
		20	133.71	3.95	0.00	5.36	0.75	0.00	213	2938	217	2941	3.69
	200 90 90	4	109.51	0.00	0.53	0.04	0.00	0.00	61	207	61	207	1.06
		8	187.28	0.59	0.59	13.14	2.41	0.00	112	917	114	918	2.23
		12	229.55	0.05	1.52	11.86	0.57	0.00	161	2080	161	2081	20.41
		16	246.32	0.87	0.20	7.53	0.76	0.00	209	4490	209	4490	11.78
	200 120 120	4	148.86	0.00	0.00	4.68	3.00	0.00	94	458	95	459	1.28
		8	255.17	1.62	0.00	11.25	3.85	0.00	188	1617	188	1617	2.19
		12	306.45	0.47	0.05	10.61	0.86	0.00	282	3337	282	3337	19.40
		16	332.52	1.35	0.17	10.59	1.25	0.00	378	7345	379	7348	22.90
	200 150 150	4	179.83	0.00	0.00	6.66	1.05	0.00	116	579	117	579	23.00
		8	306.10	0.78	0.00	8.99	1.32	0.00	234	2299	234	2299	20.05
		12	381.06	2.71	0.00	10.38	3.88	0.00	356	3993	357	3993	17.12
		16	421.40	2.08	1.07	11.77	4.36	0.00	459	10103	461	10104	15.45
			Average	1.32	0.18	8.75	2.13	0.00	156	2010	158	2010	8.93
			Maximum	4.37	1.52	13.14	7.79	0.00	459	10103	461	10104	23.00
			# Best	4	15	0	3	23					
			Total	23									

APPENDIX H

THE EFFECT OF THE NUMBER OF BOTTLE BANKS AND UNIT COST

In this appendix, costs and revenues for all possible p values are given for the problems ran50, ran100, bayg29, eil51, att48, eil76 and kroA200. These computations have been made assuming that there is no cost of locating a bottle bank. For the case with unit cost of a bottle bank, the best number of bottle banks to locate and corresponding profits are given for different cost values.

Table H-1. Revenues and costs for all possible p for the problem ran50.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	141.35	370.10	0.00	228.75	21	531.47	370.10	418.82	257.45
1	184.52	370.10	45.57	231.15	22	532.55	370.10	421.00	258.55
2	224.91	370.10	86.01	231.20	23	532.73	370.10	422.48	259.85
3	262.03	370.10	123.18	231.25	24	532.73	370.10	422.48	259.85
4	296.11	370.10	157.46	231.45	25	532.42	370.10	433.47	271.15
5	329.28	370.10	193.48	234.30	26	532.58	370.10	432.08	269.60
6	360.91	370.10	227.86	237.05	27	532.69	370.10	432.49	269.90
7	385.59	370.10	252.64	237.15	28	532.54	370.10	432.49	270.05
8	407.79	370.10	275.59	237.90	29	531.76	370.10	435.26	273.60
9	425.43	370.10	293.83	238.50	30	530.79	370.10	436.64	275.95
10	442.02	370.10	310.57	238.65	31	529.64	370.10	436.64	277.10
11	458.13	370.10	329.58	241.55	32	527.84	370.10	432.79	275.05
12	470.98	370.10	349.03	248.15	33	525.95	370.10	437.00	281.15
13	483.61	370.10	366.06	252.55	34	524.15	370.10	433.15	279.10
14	494.35	370.10	374.30	250.05	35	522.10	370.10	433.15	281.15
15	502.63	370.10	385.73	253.20	36	519.51	370.10	433.81	284.40
16	510.62	370.10	396.47	255.95	37	516.61	370.10	433.81	287.30
17	516.45	370.10	403.20	256.85	38	513.71	370.10	437.66	294.05
18	521.60	370.10	408.40	256.90	39	510.81	370.10	437.66	296.95
19	525.50	370.10	412.75	257.35	40	507.66	370.10	438.06	300.50
20	529.08	370.10	416.38	257.40					

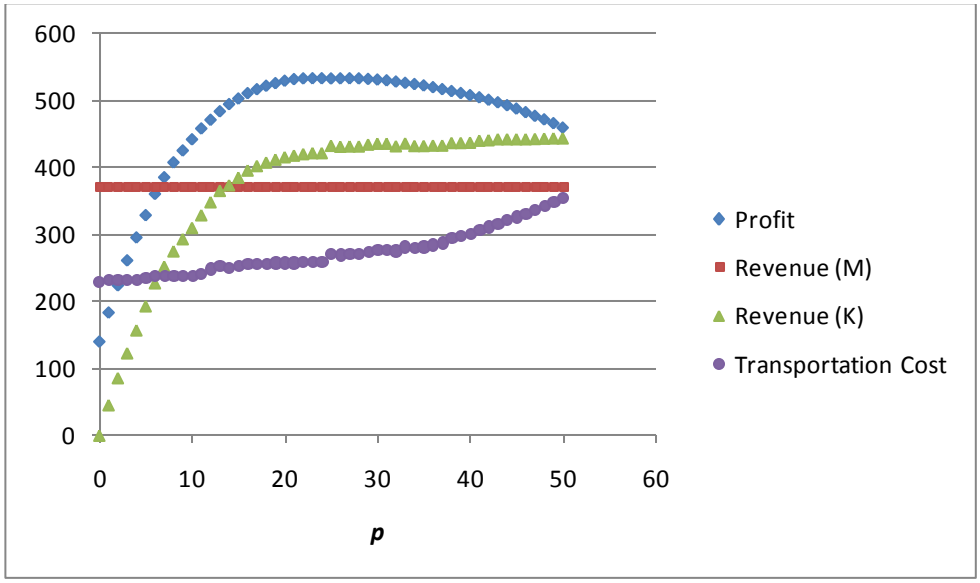

Figure H-1. Revenue and cost components for the problem ran50.

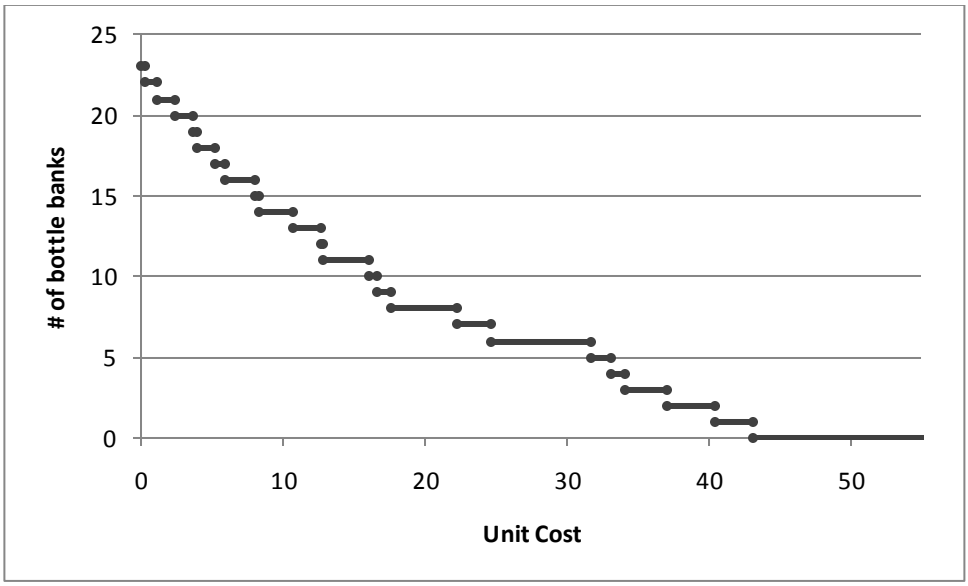

Figure H-2. The best number of bottle banks to locate for ran50.

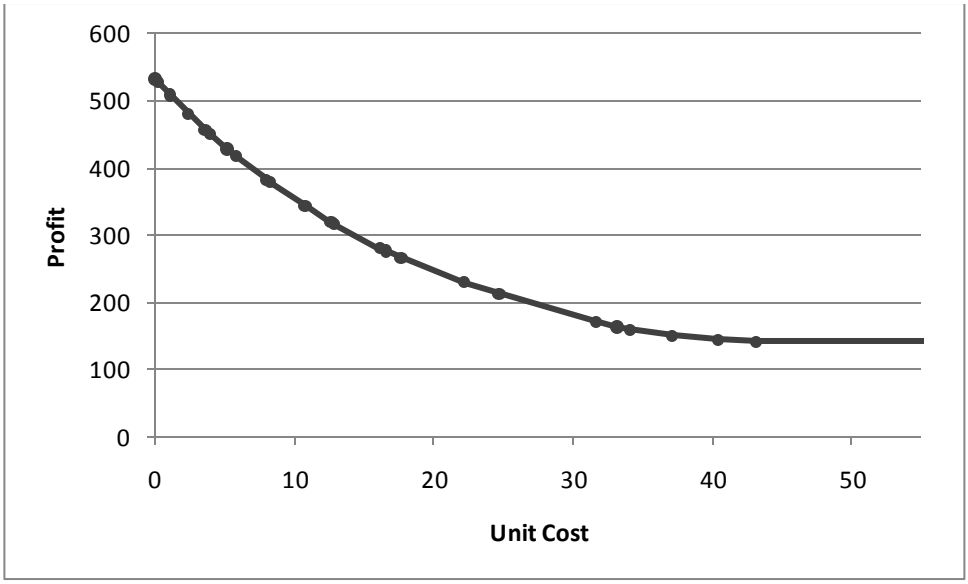

Figure H-3. Amount of profit for different unit cost, ran50.

Table H-2. Revenues and costs for all possible p for the problem ran100.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	180.70	753.50	0.00	572.80	21	965.65	753.50	820.95	608.80
1	247.13	753.50	66.73	573.10	22	975.94	753.50	831.24	608.80
2	313.10	753.50	132.80	573.20	23	981.98	753.50	837.58	609.10
3	377.54	753.50	199.54	575.50	24	985.30	753.50	840.90	609.10
4	435.52	753.50	257.52	575.50	25	988.48	753.50	844.08	609.10
5	490.17	753.50	312.37	575.70	26	989.16	753.50	850.06	614.40
6	540.81	753.50	364.21	576.90	27	991.59	753.50	847.29	609.20
7	586.14	753.50	412.34	579.70	28	992.95	753.50	852.05	612.60
8	630.85	753.50	458.95	581.60	29	992.97	753.50	848.37	608.90
9	668.55	753.50	496.45	581.40	30	995.11	753.50	854.11	612.50
10	708.82	753.50	540.72	585.40	31	994.23	753.50	845.93	605.20
11	746.48	753.50	584.48	591.50	32	995.12	753.50	864.12	622.50
12	778.13	753.50	615.73	591.10	33	995.41	753.50	857.31	615.40
13	807.35	753.50	644.95	591.10	34	995.31	753.50	857.31	615.50
14	833.76	753.50	676.86	596.60	35	995.21	753.50	857.31	615.60
15	865.23	753.50	713.93	602.20	36	994.85	753.50	846.85	605.50
16	893.65	753.50	744.85	604.70	37	994.67	753.50	853.67	612.50
17	913.79	753.50	766.09	605.80	38	994.65	753.50	864.55	623.40
18	927.96	753.50	780.26	605.80	39	994.45	753.50	864.55	623.60
19	940.60	753.50	792.90	605.80	40	994.15	753.50	864.55	623.90
20	954.21	753.50	808.51	607.80					

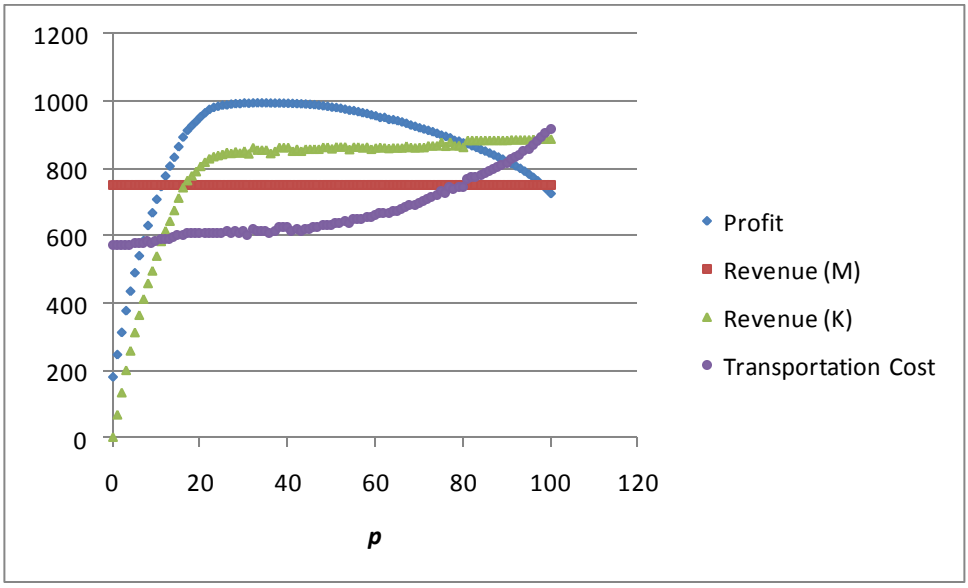

Figure H-4. Revenue and cost components for the problem ran100.

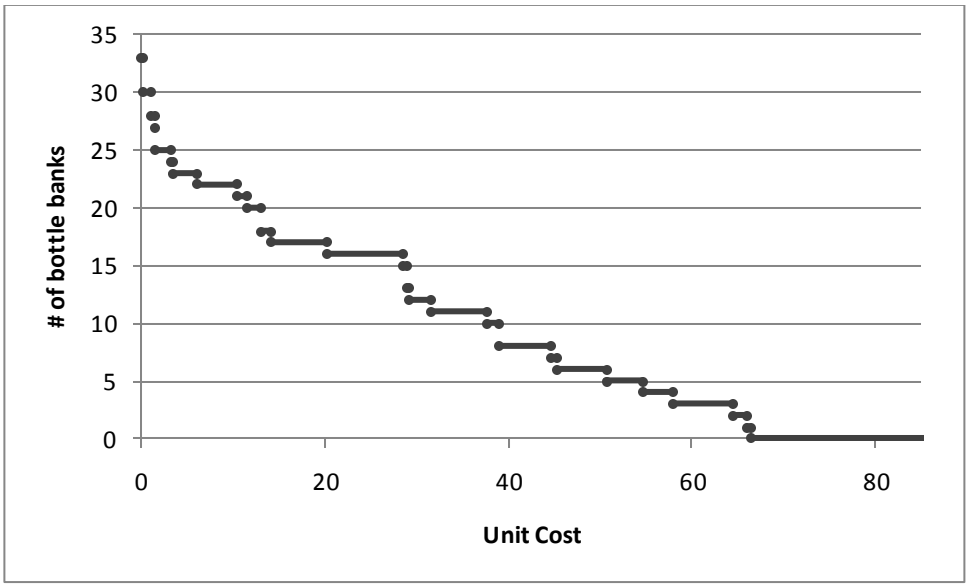

Figure H-5. The best number of bottle banks to locate for ran100.

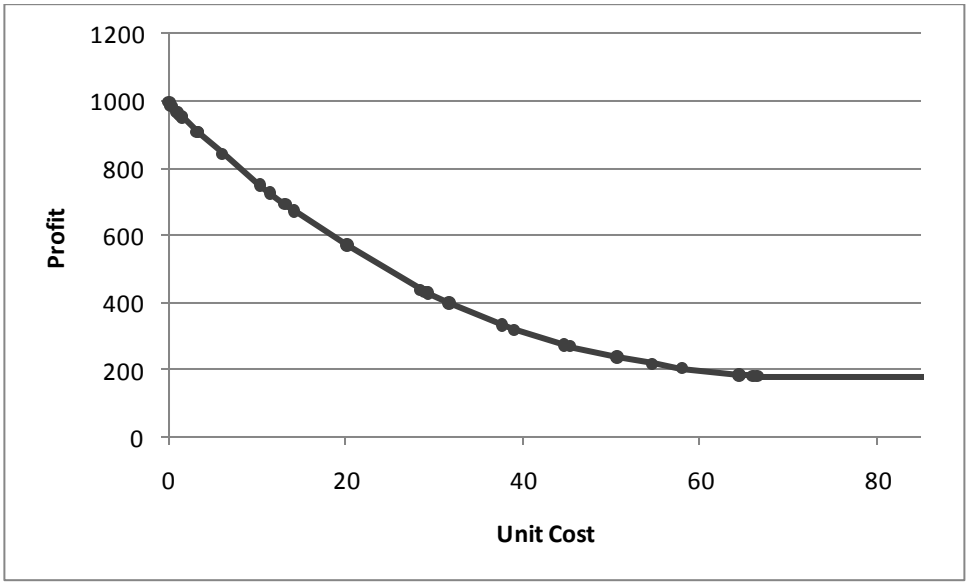

Figure H-6. Amount of profit for different unit cost, ran100.

Table H-3. Revenues and costs for all possible p for the problem bayg29-29.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	226.50	444.00	0.00	217.50	15	285.75	444.00	76.13	234.37
1	246.52	444.00	20.07	217.55	16	284.09	444.00	72.19	232.09
2	260.61	444.00	34.16	217.55	17	283.50	444.00	76.13	236.63
3	266.80	444.00	43.86	221.05	18	281.87	444.00	76.13	238.26
4	271.23	444.00	49.38	222.16	19	279.77	444.00	76.17	240.40
5	275.49	444.00	53.93	222.45	20	278.52	444.00	76.17	241.65
6	279.41	444.00	60.59	225.18	21	276.75	444.00	76.17	243.42
7	282.94	444.00	64.22	225.28	22	274.24	444.00	82.06	251.82
8	285.86	444.00	67.41	225.54	23	271.57	444.00	82.06	254.49
9	287.17	444.00	70.15	226.98	24	267.31	444.00	88.26	264.95
10	287.32	444.00	70.85	227.53	25	265.86	444.00	84.19	262.33
11	287.30	444.00	70.85	227.56	26	261.55	444.00	90.39	272.84
12	287.19	444.00	76.13	232.93	27	258.57	444.00	90.39	275.82
13	286.91	444.00	76.13	233.22	28	255.07	444.00	90.39	279.32
14	286.52	444.00	76.13	233.61	29	250.35	444.00	90.81	284.46

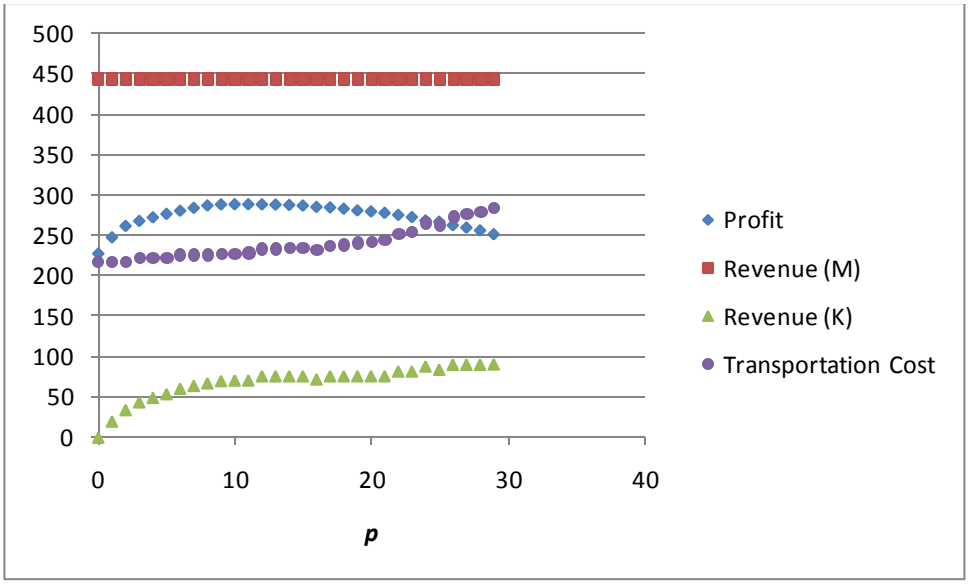

Figure H-7. Revenue and cost components for the problem bayg29-29.

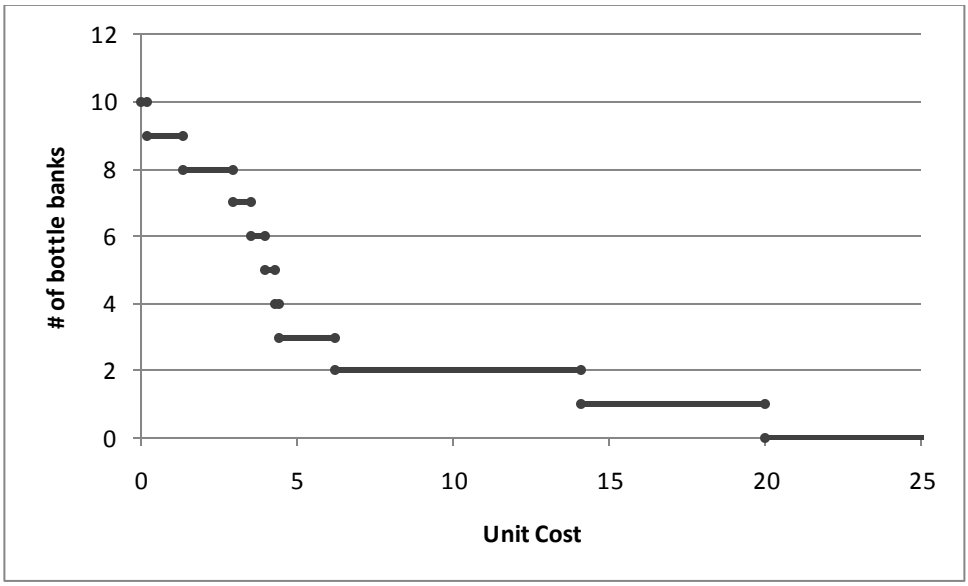

Figure H-8. The best number of bottle banks to locate for bayg29-29.

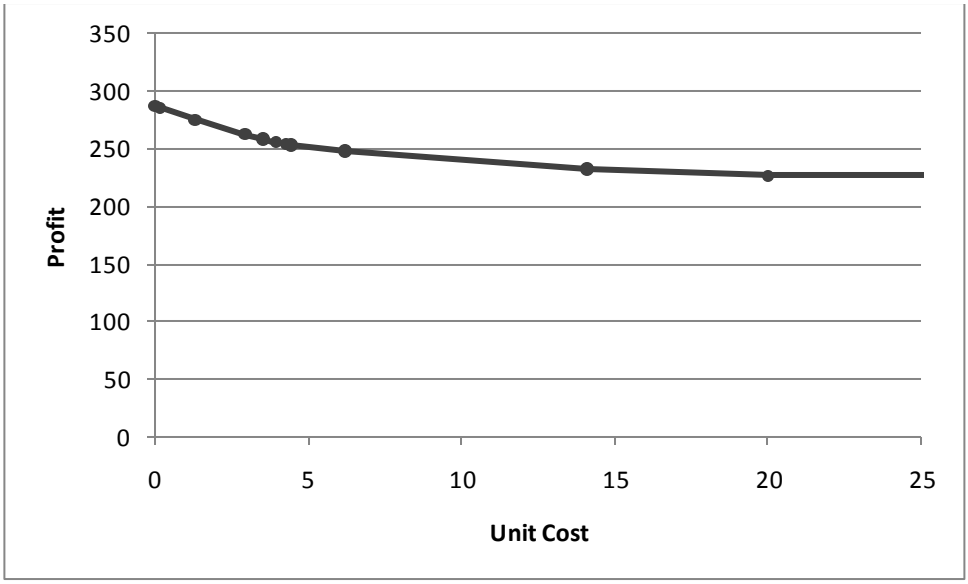

Figure H-9. Amount of profit for different unit cost, bayg29-29.

Table H-4. Revenues and costs for all possible p for the problem eil51-51.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	400.81	783.30	0.00	382.49	21	528.00	783.30	138.72	394.02
1	415.62	783.30	14.89	382.57	22	527.99	783.30	138.72	394.03
2	430.26	783.30	29.55	382.59	23	527.84	783.30	138.72	394.18
3	443.03	783.30	42.65	382.92	24	527.67	783.30	138.72	394.35
4	454.22	783.30	53.95	383.03	25	527.37	783.30	138.72	394.65
5	464.97	783.30	65.06	383.39	26	527.04	783.30	138.72	394.98
6	473.81	783.30	74.04	383.53	27	526.69	783.30	138.72	395.32
7	480.91	783.30	82.73	385.12	28	526.16	783.30	141.75	398.88
8	487.06	783.30	89.36	385.60	29	525.56	783.30	141.75	399.48
9	492.89	783.30	95.54	385.95	30	524.42	783.30	139.49	398.38
10	498.23	783.30	101.29	386.36	31	523.62	783.30	141.75	401.43
11	503.48	783.30	106.55	386.37	32	523.37	783.30	142.52	402.45
12	507.95	783.30	109.88	385.23	33	522.77	783.30	142.52	403.05
13	512.05	783.30	115.34	386.59	34	521.81	783.30	142.52	404.01
14	516.00	783.30	119.54	386.85	35	520.82	783.30	142.52	405.00
15	518.54	783.30	122.53	387.29	36	519.65	783.30	142.83	406.48
16	520.89	783.30	125.68	388.09	37	518.46	783.30	142.83	407.67
17	523.06	783.30	127.84	388.08	38	516.99	783.30	142.83	409.15
18	525.11	783.30	131.10	389.28	39	515.44	783.30	142.83	410.69
19	527.05	783.30	134.39	390.64	40	513.77	783.30	144.03	413.56
20	528.14	783.30	138.72	393.88					

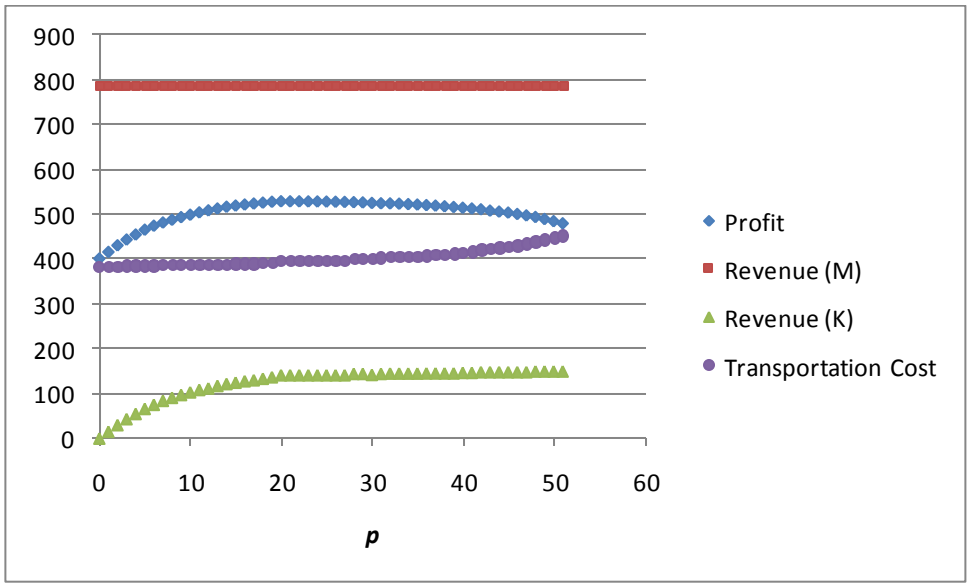

Figure H-10. Revenue and cost components for the problem eil51-51.

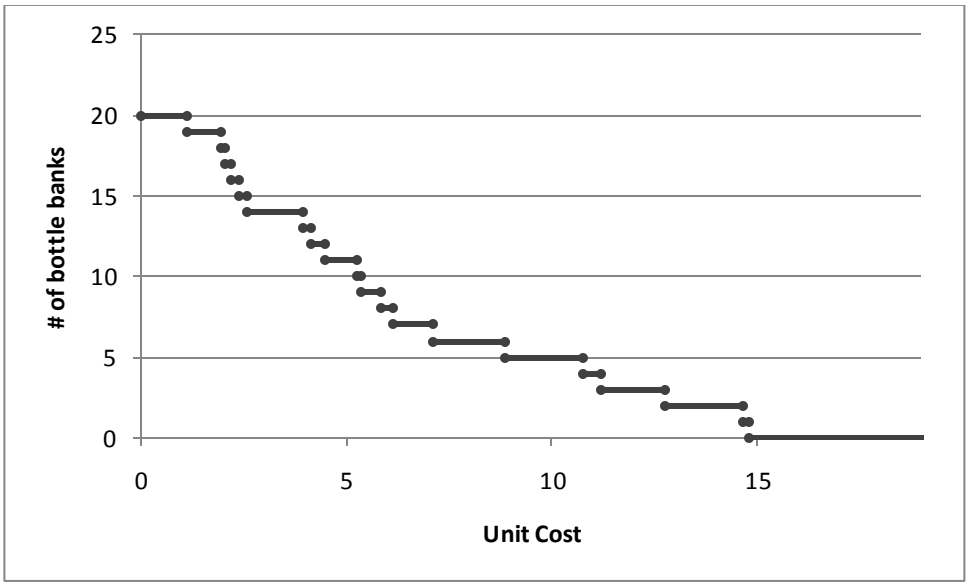

Figure H-11. The best number of bottle banks to locate for eil51-51.

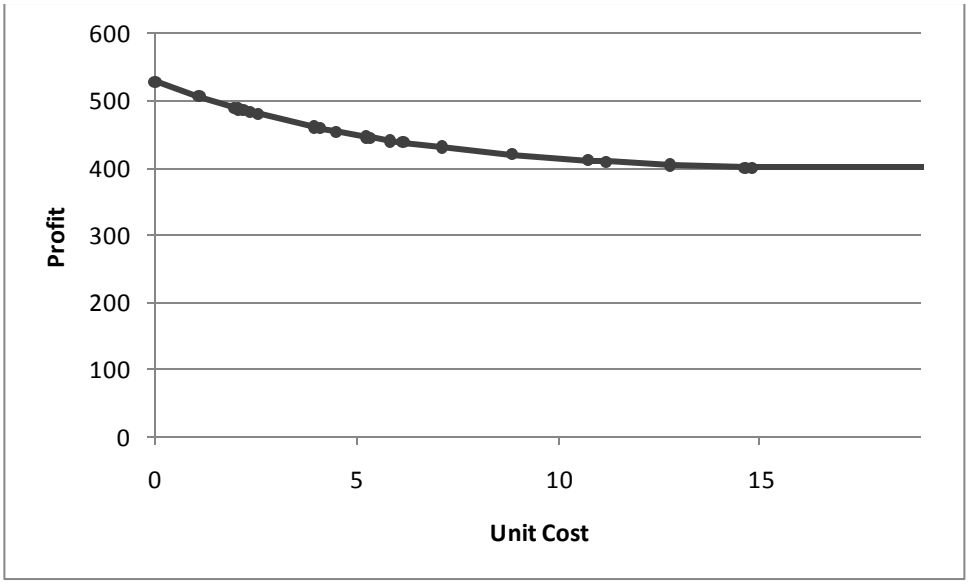

Figure H-12. Amount of profit for different unit cost, eil51-51.

Table H-5. Revenues and costs for all possible p for the problem att48-5.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	-62.26	73.30	0.00	135.56	21	114.53	73.30	251.04	209.81
1	-31.91	73.30	33.57	138.78	22	113.86	73.30	266.06	225.50
2	-4.64	73.30	66.47	144.41	23	112.02	73.30	266.06	227.34
3	20.76	73.30	91.96	144.50	24	110.32	73.30	264.63	227.61
4	41.76	73.30	121.87	153.41	25	108.48	73.30	264.63	229.45
5	59.42	73.30	139.70	153.58	26	105.63	73.30	267.14	234.81
6	76.39	73.30	159.27	156.18	27	103.34	73.30	264.63	234.59
7	89.66	73.30	174.20	157.84	28	101.97	73.30	264.63	235.96
8	100.61	73.30	185.25	157.94	29	99.41	73.30	265.19	239.08
9	109.06	73.30	195.71	159.95	30	96.56	73.30	267.70	244.44
10	116.31	73.30	218.16	175.15	31	93.51	73.30	268.93	248.72
11	119.81	73.30	222.95	176.44	32	90.16	73.30	268.93	252.07
12	121.90	73.30	224.03	175.43	33	86.45	73.30	268.93	255.78
13	123.77	73.30	222.08	171.61	34	81.26	73.30	268.93	260.97
14	124.05	73.30	230.53	179.78	35	76.04	73.30	273.55	270.81
15	124.19	73.30	242.82	191.93	36	70.85	73.30	273.55	276.00
16	124.16	73.30	244.12	193.26	37	64.84	73.30	274.79	283.25
17	123.77	73.30	246.15	195.68	38	58.51	73.30	279.82	294.61
18	122.16	73.30	248.66	199.80	39	52.62	73.30	279.82	300.50
19	119.57	73.30	248.66	202.39	40	47.02	73.30	279.82	306.10
20	116.37	73.30	251.04	207.97					

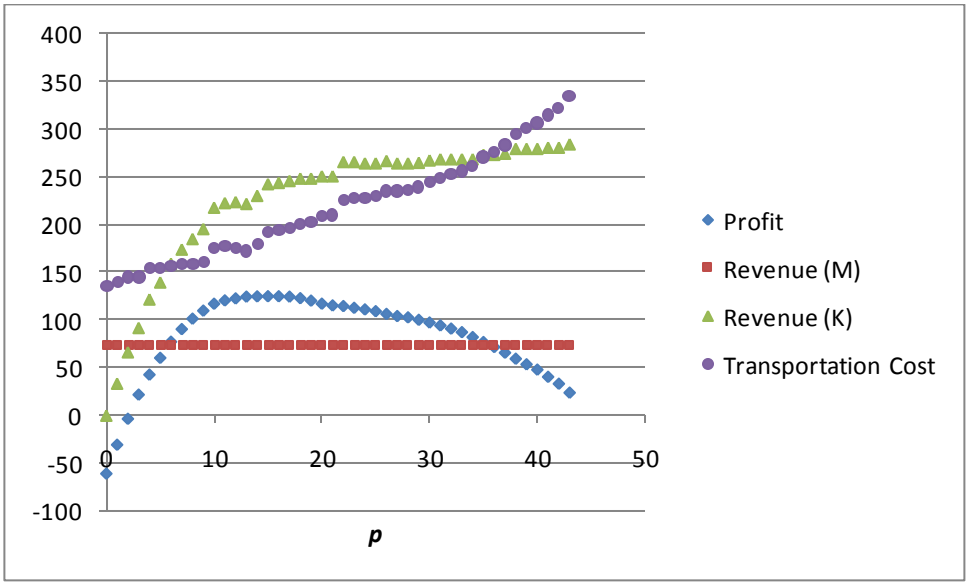

Figure H-13. Revenue and cost components for the problem att48-5.

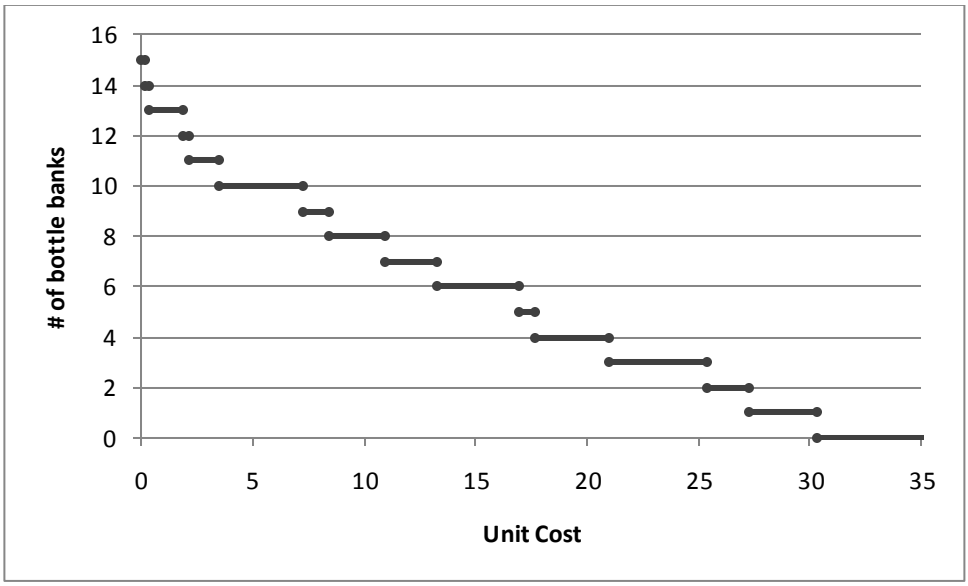

Figure H-14. The best number of bottle banks to locate for att48-5.

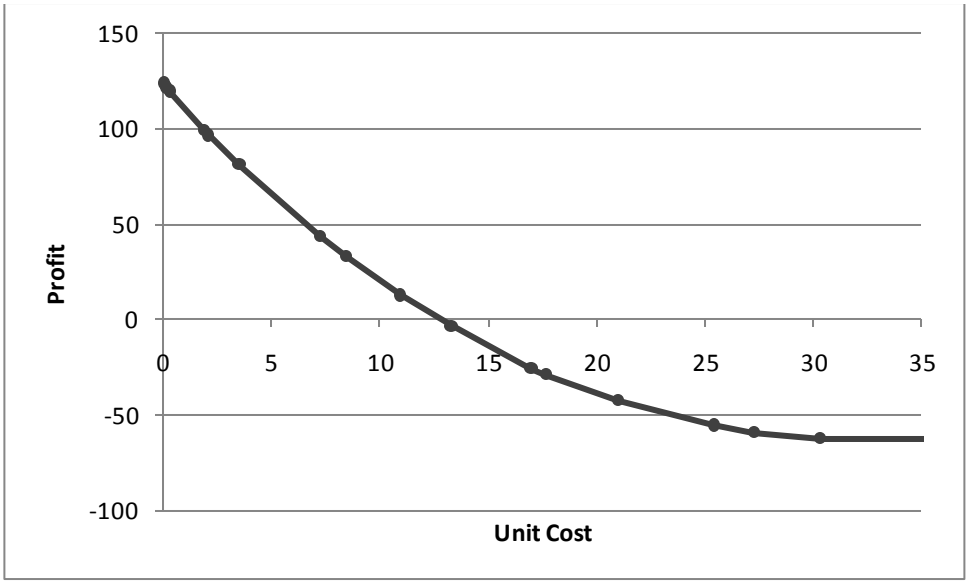

Figure H-15. Amount of profit for different unit cost, att48-5.

Table H-6. Revenues and costs for all possible p for the problem eil76-8.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	-19.75	111.20	0.00	130.95	21	350.43	111.20	489.69	250.46
1	31.90	111.20	67.44	146.74	22	351.56	111.20	507.50	267.14
2	69.67	111.20	113.18	154.71	23	351.71	111.20	507.87	267.36
3	103.73	111.20	148.01	155.48	24	351.57	111.20	509.04	268.67
4	133.43	111.20	186.08	163.85	25	351.27	111.20	524.54	284.47
5	158.34	111.20	219.87	172.73	26	351.42	111.20	524.91	284.69
6	181.90	111.20	243.66	172.96	27	351.27	111.20	526.07	286.00
7	205.25	111.20	268.79	174.74	28	350.80	111.20	526.07	286.47
8	225.63	111.20	299.46	185.03	29	349.81	111.20	526.16	287.55
9	244.33	111.20	330.07	196.94	30	348.94	111.20	553.11	315.37
10	260.90	111.20	349.16	199.46	31	347.89	111.20	553.11	316.42
11	276.76	111.20	368.54	202.98	32	346.81	111.20	553.11	317.50
12	292.11	111.20	391.14	210.23	33	345.66	111.20	553.11	318.65
13	302.05	111.20	401.30	210.45	34	344.43	111.20	553.11	319.88
14	311.88	111.20	421.72	221.04	35	342.45	111.20	553.11	321.86
15	320.79	111.20	438.08	228.49	36	339.98	111.20	554.05	325.27
16	329.68	111.20	451.52	233.04	37	337.50	111.20	554.76	328.46
17	337.64	111.20	460.40	233.96	38	334.70	111.20	554.76	331.26
18	341.84	111.20	468.79	238.15	39	331.78	111.20	556.11	335.53
19	345.29	111.20	479.58	245.49	40	328.66	111.20	568.15	350.69
20	347.99	111.20	487.12	250.33					

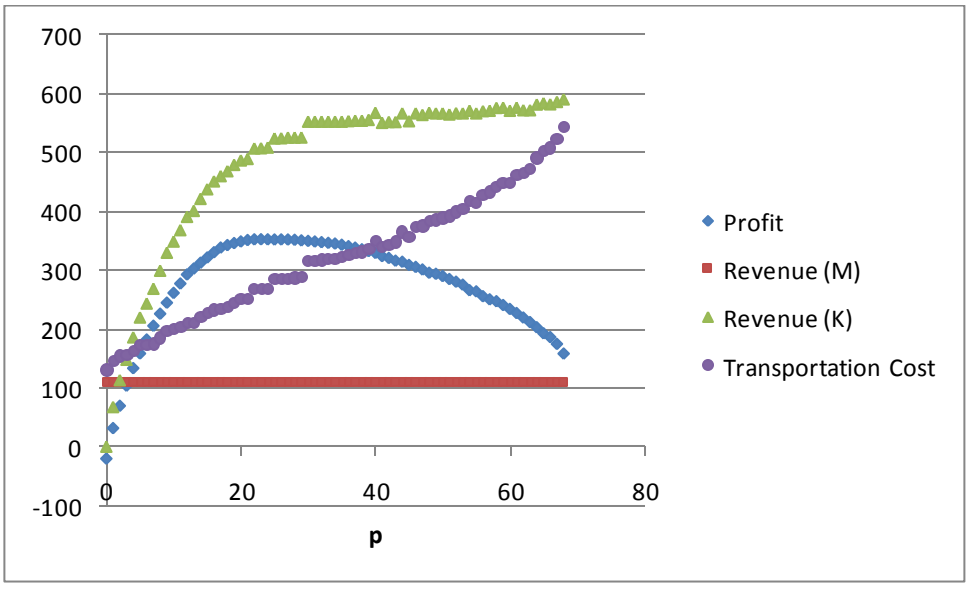

Figure H-16. Revenue and cost components for the problem eil76-8.

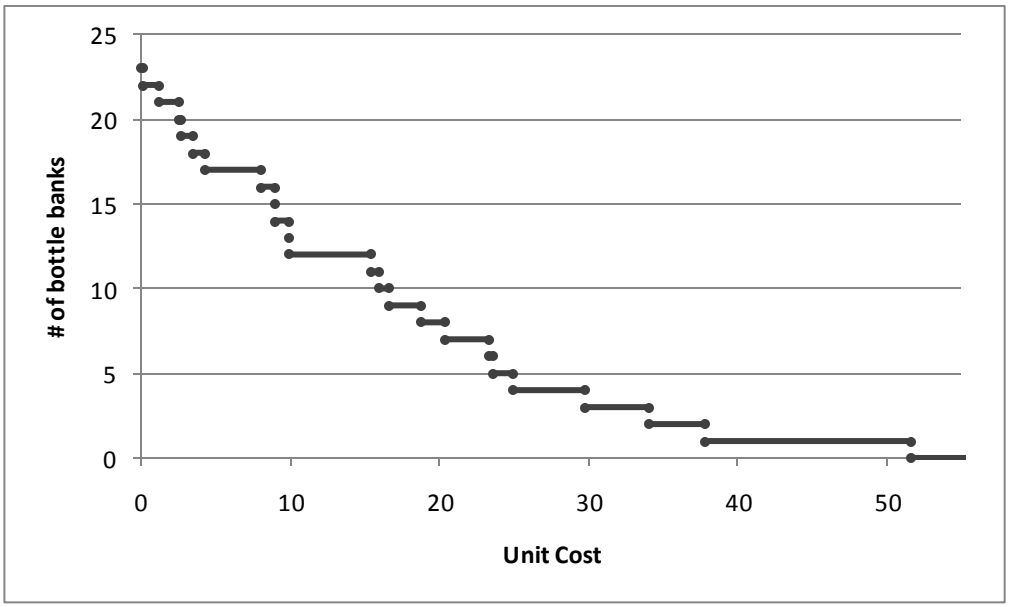

Figure H-17. The best number of bottle banks to locate for eil76-8.

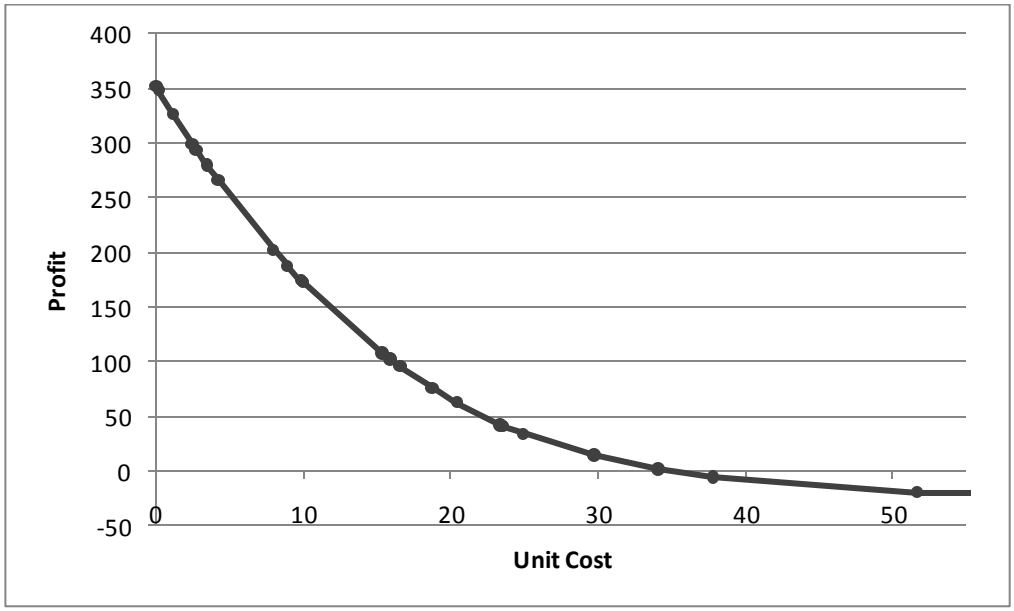

Figure H-18. Amount of profit for different unit cost, eil76-8.

Table H-7. Revenues and costs for all possible p for the problem kroA200.

p	Profit	Revenue (M)	Revenue (K)	Transportation Cost	p	Profit	Revenue (M)	Revenue (K)	Transportation Cost
0	1436.40	2936.40	0.00	1500.00	21	1687.98	2936.40	255.53	1503.94
1	1471.10	2936.40	34.75	1500.05	22	1687.98	2936.40	255.53	1503.94
2	1498.77	2936.40	62.68	1500.31	23	1687.83	2936.40	255.53	1504.10
3	1523.13	2936.40	87.50	1500.77	24	1687.62	2936.40	255.53	1504.30
4	1545.91	2936.40	111.51	1502.00	25	1687.37	2936.40	255.53	1504.56
5	1568.66	2936.40	134.25	1502.00	26	1687.11	2936.40	255.53	1504.82
6	1588.16	2936.40	154.48	1502.71	27	1686.85	2936.40	255.53	1505.07
7	1607.14	2936.40	173.81	1503.07	28	1686.60	2936.40	255.53	1505.33
8	1623.68	2936.40	190.61	1503.33	29	1686.34	2936.40	255.53	1505.58
9	1636.29	2936.40	203.22	1503.33	30	1686.09	2936.40	255.53	1505.84
10	1649.58	2936.40	216.92	1503.74	31	1685.68	2936.40	255.53	1506.25
11	1660.56	2936.40	227.95	1503.79	32	1685.27	2936.40	255.53	1506.66
12	1665.95	2936.40	233.04	1503.48	33	1684.81	2936.40	255.53	1507.12
13	1671.74	2936.40	238.87	1503.53	34	1684.29	2936.40	255.53	1507.63
14	1676.24	2936.40	243.58	1503.74	35	1683.83	2936.40	255.53	1508.09
15	1679.86	2936.40	247.35	1503.89	36	1683.27	2936.40	255.53	1508.66
16	1682.72	2936.40	249.95	1503.64	37	1682.28	2936.40	255.71	1509.84
17	1685.42	2936.40	252.66	1503.64	38	1680.84	2936.40	255.71	1511.27
18	1687.79	2936.40	255.03	1503.64	39	1679.36	2936.40	255.71	1512.76
19	1687.95	2936.40	255.44	1503.89	40	1676.74	2936.40	257.24	1516.90
20	1687.98	2936.40	255.53	1503.94					

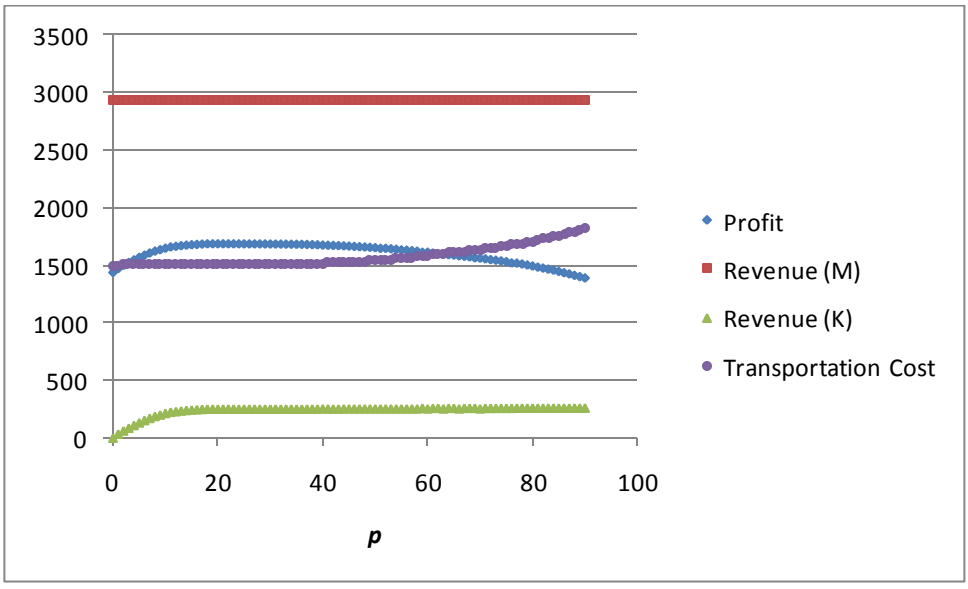

Figure H-19. Revenue and cost components for the problem kroA200.

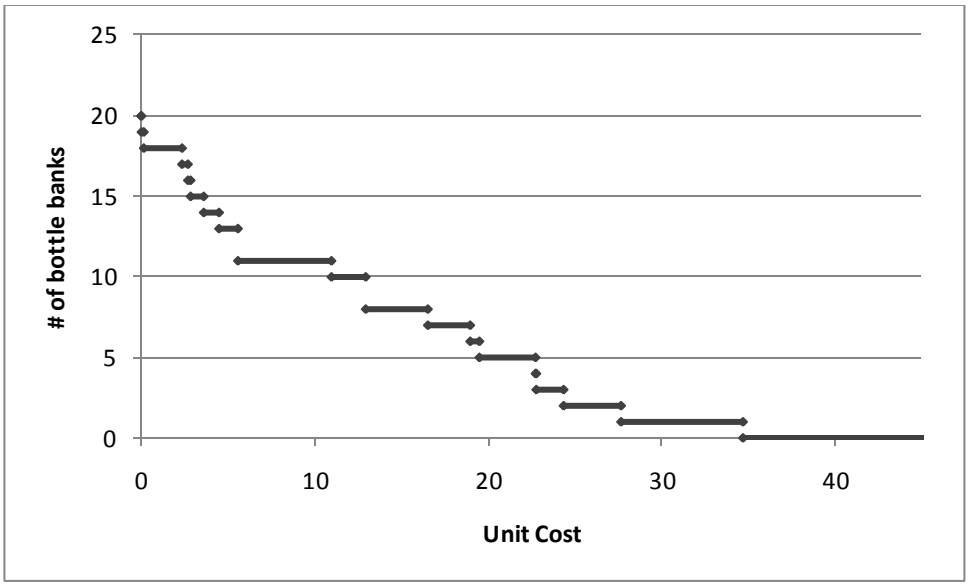

Figure H-20. The best number of bottle banks to locate for kroA200.

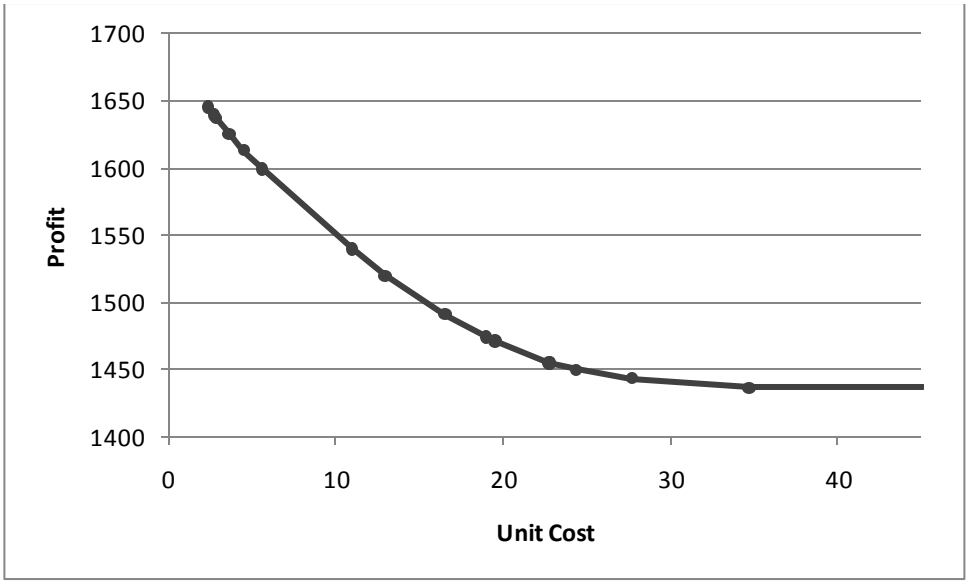

Figure H-21. Amount of profit for different unit cost, kroA200.