SOLUTION PROCEDURES FOR THE RECTILINEAR DISTANCE SINGLE SOURCE CAPACITATED MULTI-FACILITY WEBER PROBLEM

(TEK KAYNAKLI SINIRLI SIĞALI ÇOK TESİSLİ DİK DOĞRUSAL UZAKLIKLI WEBER PROBLEMİ İÇİN ÇÖZÜM YÖNTEMLERİ)

by

M. Emre DEMİRCİOĞLU, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

INDUSTRIAL ENGINEERING

in the

INSTITUTE OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

January, 2012

SOLUTION PROCEDURES FOR THE RECTILINEAR DISTANCE SINGLE SOURCE CAPACITATED MULTI-FACILITY WEBER PROBLEM

(TEK KAYNAKLI SINIRLI SIĞALI ÇOK TESİSLİ DİK DOĞRUSAL UZAKLIKLI WEBER PROBLEMİ İÇİN ÇÖZÜM YÖNTEMLERİ)

by

M. Emre DEMİRCİOĞLU, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Date of Submission : Jan 6, 2012

Date of Defense Examination : Jan 27, 2012

Supervisor : Assoc. Prof. Dr. Temel ÖNCAN

Committee Members : Asst. Prof. Dr. M. Ebru ANGÜN

Asst. Prof. Dr. B. Atay ÖZGÖVDE

ACKNOWLEDGEMENT

I would like to wish to express my sincere thanks to Assoc. Prof. Temel Öncan for his supervision, support, guidance of this study and encouraging.

I would like to express my thanks to committee members, Asst. Prof. Dr. M. Ebru Angün, and Asst. Prof. Dr. B. Atay Özgövde, for their comments and words of encouragements.

I would like to thank to my family and my friends for their encouragement, love and support.

M. Emre DEMİRCİOĞLU
January, 2012

TABLE OF CONTENTS

LIST OF SYMBOLS

ACS : Ant Colony System
AI-VLSN : Allocation Improvement Very Large Scale Neighbourhood
ALA : Alternate Location-Allocation
ATL : Alternate Transportation-Location
BB : Branch-and-Bound
BP : Bilinear Programming
CAP : Covering Assignment Problem
CBADR : Cell-Based Approach with Dynamic Radius
CBAFR : Cell-Based Approach with Fixed Radius
CLSN : Compromised Large-Scale Neighbourhood
CMST : Capacitated Minimum Spanning Tree
CMWP : Capacitated Multi-Facility Weber Problem
DA : Discrete Approximation
DAP : Discrete Approximation Problem
DP : Dynamic Programming
FC-VLSN : Feasibility Chasing Very Large Scale Neighbourhood
FLA : Facility Location-Allocation
GA : Genetic Algorithm
GAP : Generalized Assignment Problem
GRASP : Greedy Randomized Adaptive Search Procedure
HEM : Hybrid Evolutionary Method
KP : Knapsack Problem
LAP : Location-Allocation Problem
LP : Linear Programming
LR : Lagrangian Relaxation
LS : Local Search
MCFLP : Multi-Commodity Capacitated Facility Location Problem

MCMWP : Multi-Commodity Capacitated Multi-Facility Weber Problem
MILP : Mixed Integer Linear Programming
MRGAP : Multi-Resource Generalized Assignment Problem
MWP : Multi-Facility Weber Problem
QAP : Quadratic Assignment Problem
RBM : Recency Based Memory
RE : Random Ejection
RLT : Reformulation-Linearization Technique
RR : Region-Rejection
RRA : Region-Rejection Approach
RRADR : Region-Rejection Approach with Dynamic Radius
RRAFR : Region-Rejection Approach with Fixed Radius
SOM : Self-Organizing Map
SSCFLP : Single Source Capacitated Facility Location Problem
SSCMWP : Single Source Capacitated Multi-facility Weber Problem
SSTP : Single Source Transportation Problem
TA : Threshold Accepting
TP : Transportation Problem
TS : Tabu Search
TSP : Travelling Salesman Problem
VLSN : Very Large Scale Neighbourhood
VNS : Variable Neighbourhood Search
VRP : Vehicle Routing Problem
WP : Weber Problem

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The facility location-allocation problem is an important topic that arises in many practical settings. These applications arise in various areas such as transportation, distribution, production, supply chain decisions and telecommunication. As a result, the studies on the facility location problems are steadily increasing in the literature.

In this work, we have concentrated on the Single Source Capacitated Multi-Facility Weber Problem (SSCMWP) which aims to tackle the optimal location of a number of facilities such as plants, warehouses or concentrators that will serve a set of customers with known locations under single source assumption. The SSCMWP is a non-convex optimization problem and hence difficult to solve.

The SSCMWP can be considered as the Capacitated Multi-facility Weber Problem (CMWP) when the customers are not forced to serve from exactly one facility and the CMWP becomes Multi-Facility Weber Problem (MWP) when the capacity restrictions on facilities are ignored. Furthermore, the MWP reduces to the Weber Problem (WP) when the objective is to find optimal location for a single facility that will serve a set of customers. It is well-known that the WP is solved by Weiszfeld's algorithm to optimality.

Previous studies on the WP and its extensions have shown that the optimal facility locations lie on the points obtained by the intersection of vertical and horizontal lines drawn by passing through the customer locations. Keeping in mind this result, we have presented Discrete Approximation Problem (DAP) which can yield the optimal solution of the rectilinear distance SSCMWP. One weakness of the DAP approach is its drastic CPU time requirement. Especially for large instances, it may not be possible to find the optimal solution of the rectilinear distance SSCMWP by solving DAP to optimality.

Hence, this fact is one of our motivations to construct efficient and accurate heuristics for the rectilinear distance SSCMWP.

The first heuristic that we consider in this study is an Alternate Location Allocation type (ALA) heuristic which is inspired from the seminal work of Cooper (1972). We have adopted his approach in order to compare it with other heuristic algorithms which we propose for the rectilinear distance SSCMWP. The ALA heuristic consists of two phases: allocation phase and location phase. In the allocation phase, we have employed a commercial MILP solver (i.e. CPLEX) to solve the Single Source Transportation Problem (SSTP). In the location phase, we have performed the Weiszfeld's algorithm. We have observed that especially for the large SSTP instances, running the ALA type heuristic becomes a computationally painstaking task. Thus, we have resorted to improve the allocation phase of the ALA type heuristic by heuristically solving the SSTP either by a Very Large Scale Neighbourhood (VLSN) search algorithm or by a Tabu Search (TS) algorithm.

The second heuristic that we suggest incorporates a VLSN search algorithm to solve the SSTP. That is to say, the VLSN search algorithm is performed as a subprocedure within the ALA heuristic in order to heuristically solve the SSTP. The VLSN search heuristic is constructed by using multi-exchange moves with some ejection rules. We have designed two versions of the VLSN search algorithms: feasible chasing and allocation improvement. As ejection rules, we have employed only two simple rules: Random Ejection (RE) rule and Recency Based Memory (RBM) rule.

The third heuristic that we propose includes a TS algorithm to solve the SSTP. Within this algorithm, we have devised swap moves. Furthermore, we have devised two versions of the TS algorithm. In the first version, we allow infeasible moves while in the second version we do not allow infeasible moves.

By performing computational experiments on test instances obtained from the OR-library, we have observed that both VLSN search and TS algorithms yield substantial improvements over the ALA type heuristic. Especially we should note that the

proposed VLSN search algorithm yields significantly better results especially in reasonable CPU times.

RESUME

Les modèles de localisation allocation sont des sujets importants qui se posent dans de nombreux contextes pratiques. Ces applications peuvent être employées dans de divers domaines comme le transport, la distribution, la production des décisions d'inventaire et de télécommunication. Ainsi, les recherches pour les problèmes de la localisation et allocation augmentent de plus en plus dans la littérature.

Dans ce travail, nous nous sommes concentrés sur le Problème de Weber multi-service avec la capacité qualifiée par la source unique (en Anglais : SSCMWP) qui vise à aborder l'endroit optimal d'un certain nombre d'équipements tels que les usines, les entrepôts ou concentrateurs qui serviront un ensemble de clients avec des lieux connus sous l'hypothèse de source unique. Le SSCMWP appartient à l'optimisation non-convexe et donc c'est un problème difficile á résoudre.

Le SSCMWP peut être considéré le problème de Weber multi-service avec capacité (CMWP) lorsque les clients ne sont pas obligés de servir d'exactement un établissement et CMWP devient le Problème de Weber multi-service (MWP) lorsque les restrictions de capacité sur les établissements sont ignorées. Par ailleurs, le MWP sera réduit au Problème de Weber (WP) quand l'objectif est de trouver l'endroit optimal pour un seul établissement qui servira un ensemble de clients. Il est bien connu que la WP peut être résolu par l'algorithme de Weiszfeld de façon optimale.

Les études antérieures sur la WP et ses extensions ont montré que les endroits optimaux se situent sur les points obtenus par l'intersection des lignes verticales et horizontales tracées en passant par les sites des clients. De plus, nous avons présenté problème d'approximation discrète (DAP) qui peut rapporter la solution optimale pour l'SSCMWP rectiligne. Une faiblesse de l'approche DAP est son exigence drastique du temps CPU. Surtout pour les instances grandes, il peut être impossible de trouver la solution

optimale de la distance rectiligne de l'SSCMWP en résolvant DAP à l'optimalité. Ainsi, ce fait est une de nos motivations pour construire des heuristiques efficaces et précis pour la distance rectiligne de SSCMWP.

La première heuristique que nous considérons dans cette étude est l'heuristique de type Localisation Allocation Alterné (ALA) qui est inspiré du travail séminal de Cooper (1972). Nous avons adopté son approche pour le cas SSCMWP afin de le comparer avec d'autres algorithmes heuristiques que nous proposons pour la distance rectiligne SSCMWP. ALA heuristique se compose de deux phases: la phase d'allocation et la phase de localisation. Dans la phase d'allocation, nous avons utilisé CPLEX, un solutionneur commercial pour résoudre le Problème de Transport par un Source Unique (SSTP). Dans la phase de localisation, nous avons effectué l'algorithme de Weiszfeld. Nous avons observé que surtout pour les grandes instances, la phase d'allocation dans l'heuristique d'ALA type devient une tâche de calcul minutieux. Ainsi, nous avons recouru à améliorer la phase d'allocation de l'heuristique de type heuristique ALA pour la résolution du SSTP, par un algorithme de recherche du Voisinage de Très Grande Echelle (VLSN) ou par un algorithme de Tabu Recherche (TS).

La deuxième heuristique que nous proposons pour résoudre le SSTP est l'algorithme de recherche de VLSN. L'algorithme de recherche de VLSN est réalisé comme une sous-procédure dans l'ALA, afin de résoudre heuristiquement le SSTP. L'heuristique de recherche de VLSN est construite en employant plusieurs multi-échanges des mouvements qui se déplace avec certaines règles d'éjection. Nous avons conçu deux versions des algorithmes de recherche de VLSN: chasser par VLSN et l'amélioration d'allocation. Comme les règles d'éjection, nous avons employé seulement deux règles simples: la règle d'Ejection Aléatoire (RE) et la règle de la Mémoire Récence Basé (RBM).

Le troisième heuristique que nous suggérons est un algorithme de TS qui est l'une des techniques bien connues dans la recherche locale en optimisation combinatoire. Dans cet algorithme, nous avons des mouvements d'échange. Par ailleurs, nous avons conçu deux versions de l'algorithme de TS. Dans la première version, nous permettons des

mouvements infaisables alors que dans la deuxième version nous ne permettons pas des mouvements infaisables.

En effectuant des expériences sur les instances de la bibliothèque d'OR, nous avons observé que les algorithmes de VLSN et de TS rapportent des améliorations substantielles au-dessus de l'heuristique de type ALA. Particulièrement il faut noter que l'algorithme de recherche de VLSN donne des résultats significativement meilleurs que les autres surtout dans un temps de CPU raisonnable.

ÖZET

Tesis yerleştirme ve atama problemi, birçok alanda pratik uygulaması olan önemli bir konudur. Bu uygulamalar; ulaşım, dağıtım, üretim, tedarik zinciri yönetimi ve telekomünikasyon gibi alanlarda görülebilir. Bu nedenle, yazında tesis planlama problemleri için araştırmalar giderek artmaktadır.

Bu çalışmada, birçok farklı tipteki tesis planlama problemleri arasından, tek kaynaktan tedarik edildiği varsayımı altında, yerleri bilinen istemciler için depo, işletme gibi birden fazla tesisin en iyi yerini bulmayı amaçlayan Tek Kaynaklı Sınırlı Sığalı Çok Tesisli Weber Problemi (TKSÇWP) ele alındı. TKSÇWP bir dış bükey olmayan eniyileme problemi olmasından ötürü, çözümü zor bir problemdir.

İstemcilerin tek bir tesisten hizmet almak zorunda olmadığı durumda problem, Sınırlı Sığalı Çok Tesisli Weber Problemi (SÇWP) olarak tanımlanırken; tesisler üzerindeki sığa kısıtları dikkate alınmadığı durumda Çok Tesisli Weber Problemine (ÇWP) indirgenmektedir. ÇWP ise birçok istemcinin tek bir tesis tarafından hizmet aldığı ve bu tesisin en iyi yerinin arandığı durumda Weber Problemine (WP) indirgenir. Bilindiği üzere Weiszfeld yöntemi, Weber Probleminin en iyi çözümünü bulabilmektedir.

WP ile ilgili geçmişteki çalışmalar, en iyi tesis yerlerinin, alıcıların bulunduğu noktalardan geçen yatay ve dikey doğruların kesiştiği noktalar üzerinde olduğunu göstermiştir. Bu sebeple, dik uzaklıklı TKSÇWP'nin en iyi çözümünü bulabilmek için Kesikli Yaklaşım Problemi sezgiselini (KYP) çözmek yeterli olacaktır. Uzun süreye duyulan gereksinim, KYP sezgiselin zayıf bir yanıdır. Özellikle büyük boyutlu örnekler için dik uzaklıklı TKSÇWP probleminin KYP sezgiseli yardımıyla en iyi çözümünü

bulmak mümkün olmayabilir. Dolayısıyla, bu gerçek bizi, dik uzaklı TKSÇWP problemi için daha etkili ve verimli sezgiselleri tasarlamaya yöneltti.

Bu çalışmada geliştirdiğimiz ilk sezgisel yöntem, Cooper'ın çalışmasından yararlanarak önerdiğimiz Almaşık Yerleştirme – Atama (AYA) tipinde bir sezgisel dizgi işlemidir. Dik uzaklıklı TKSÇWP için önerilen diğer dizgi işlemlerle karşılaştırabilmek adına, bu basit yöntemi TKSÇWP için uyarladık. AYA sezgiseli iki aşama içermektedir: atama aşaması ve yerleştirme aşaması. Atama aşamasında, Tek Kaynaklı Taşıma Problemini (TKTP) çözebilmek adına CPLEX isimli ticari bir çözücü kullandık. Yerleştirme aşamasında ise, Weiszfeld yöntemini kullandık. Özellikle büyük boyutlu TKTP örnekleri için AYA tipinde bir sezgisel dizgi işleminin oldukça zahmetli bir iş olduğunu gözlemledik. Bunun üzerine, AYA tipinde bir sezgisel dizgi işleminin atama aşamasındaki TKTP problemini sezgisel olarak çözmek için Çok Büyük Ölçekli Komşuluk (ÇBÖK) arama dizgi işlemi ve Tabu Arama (TA) dizgi işlemi gibi yöntemlere başvurduk.

ÇBÖK arama dizgi işlemi, TKTP problemini çözmek için düşündüğümüz ikinci sezgiseldir. ÇBÖK arama dizgi işlemi AYA sezgiselinin içinde yer alan TKTP problemini sezgisel olarak çözmek için bir alt yöntem olarak çalışmaktadır. ÇBÖK arama dizgi işlemi, çeşitli eşleme kurallarıyla birlikte çoklu yapısı kullanılarak geliştirildi. ÇBÖK arama dizgi işleminin iki farklı türü geliştirildi: olurlu yapma ve atama iyileştirme. Eşleme kuralları olarak iki basit kural kullandık: Rassal Eşleme (RE) kuralı ve Kısa Dönemli Bellek (KDB) kuralı.

Çalışmamızda önerdiğimiz üçüncü sezgisel yöntem, birleşi eniyilemesinde bilinen yerel arama yöntemlerinden biri olan TA dizgi işlemidir. Bu dizgi işlemde, ikili değişimler kullanıldı. Bununla birlikte, TA dizgi işleminin iki farklı sürümü geliştirildi. İlk sürümünde, olursuz hareketlere izin verirken, ikinci sürümünde olursuz hareketlere izin verilmedi.

OR-kütüphanesinden alınan deneme örneklerindeki sayısal deneylerde, gerek ÇBÖK arama, gerekse TA dizgi işlemlerinin AYA tipinde bir sezgisel dizgi işleminde olumlu sonuçlar verdiğini ve iyileştirme sağladığını gözlemledik. Özellikle ÇBÖK arama dizgi işleminin hissedilir oranda kabul edilebilir bir sürede sonuç verdiğini söyleyebiliriz.

1. INTRODUCTION

Facility location problems deal with the question of where to locate a single object or a set of objects. Broadly speaking, there are basically two groups of facility location problems: discrete location problems and continuous location problems. In many real world applications, the facilities to be located can be warehouses, factories, depots, service centers, concentrators, distribution centers, antennas or retailers. Furthermore, the existing facilities are usually customers which require the collection or distribution of a single product or a set of products.

Studies on the facility location problems date back to the seminal work by Weber [1] who has defined the well-known Weber Problem (WP) which deals with the optimum location of a single facility on the Euclidean space such that the sum of the distances from this facility to the existing customers with fixed locations is minimized. In other words, the WP tries to find the location of a single facility in the plane, such that given a weight q_j for each fixed customer $j = 1,..., N$, the total distance from customers to the facility, i.e. $\sum_{j=1}^{J} d(\mathrm{x}, \mathrm{a}_j)$, is minimized. Here, x is the unknown location of facility, a_j is the fixed location of customer j and $d(\mathrm{x}, \mathrm{a}_j)$ denotes the distance between the facility and customer j.

Since its introduction, the WP and its variants have been addressed by several researchers. Cooper [2] has published his pioneer work on the Multi-facility Weber Problem (MWP) which consists of the location of M uncapacitated facilities in the plane to satisfy the demand of N customers at minimum total transportation cost. Yet a further extension of the MWP is the Capacitated Multi-Facility Weber Problem (CMWP). Given the locations of N customers and their fixed demands, the CMWP deals with locating M capacitated facilities in the plane and satisfying the demands of

N customers at minimum total transportation cost which is proportional to the distances between them. The CMWP is shown to be NP-hard by Sherali and Nordai [3] even if the customers are located on a line. Observe that the CMWP becomes the MWP when the capacity restrictions on facilities are ignored. Furthermore note also that, the MWP reduces to the WP when we consider the optimal location of a single facility. However, in an optimal solution of the MWP each customer is served from the nearest facility which is not true for the more restricted CMWP because the facilities have limited capacities and the demand of customers can be satisfied from different facilities. As a further extension of the CMWP, we can mention the Single Source Capacitated Multi-facility Weber Problem (SSCMWP). The SSCMWP is concerned with the optimal location of M capacitated facilities in the plane and satisfying the demand of N customers at minimum total transportation cost such that each customer satisfies all his demand from exactly one facility. The *single source* restriction in the SSCMWP may arise in several real life problem contexts such as custom regulations and trade agreements. The single source restriction is also incorporated in several other problems in the literature. Two examples for this are the Single Source Transportation Problem (SSTP) and the Covering Assignment Problem (CAP). The SSTP incorporates the Transportation Problem (TP) together with the additional requirement that the entire demand at each customer is supplied from an exactly one supplier.

On the other hand, the CAP has also a similar structure to the SSTP. The CAP is first studied by Foulds and Wilson [4]. Both the CAP and the SSTP can be considered as special cases of the Generalized Assignment Problem (GAP). As an interesting real-world application of the CAP, Foulds and Wilson [4] have mentioned a problem which arises in the New Zealand dairy industry, where the milk demand of dairy companies is supplied from different farms. In this application the CAP consists of finding the optimal allocation of farms to factories such that each farm supplies exactly one factory. There may exist various extensions of the CMWP. One such extension is the Multi-Commodity Capacitated Multi-Facility Weber Problem (MCMWP) which has been studied by Akyüz et al. ([5, 6, 7, 8]). Moreover, all these problems, i.e., WP, MWP, CMWP, MCMWP, SSCMWP, can be categorized according to their distance functions. The most frequently used distance functions are given in Table 1.1., where $\left(x_{i1,} x_{i2}\right)$

stands for the coordinates of the facility i and (a_{j1}, a_{j2}) denotes the location of customer j.

Table 1.1. Formula for the CMWP Distance Functions Used in the Literature

Distance Functions	**Formula**
Rectilinear or Manhattan Distance (l_1 Distance)	$\lvert x_{i1} - a_{j1} \rvert + \lvert x_{i2} - a_{j2} \rvert$
Euclidean Distance (l_2 Distance)	$\left[(x_{i1} - a_{j1})^2 + (x_{i2} - a_{j2})^2 \right]^{1/2}$
Squared Euclidean Distance ($l_2{}^2$ Distance)	$(x_{i1} - a_{j1})^2 + (x_{i2} - a_{j2})^2$
l_p Distance	$\left[\lvert x_{i1} - a_{j1} \rvert^p + \lvert x_{i2} - a_{j2} \rvert^p \right]^{1/p}$
l_p^q Distance	$\left[\lvert x_{i1} - a_{j1} \rvert^p + \lvert x_{i2} - a_{j2} \rvert^p \right]^{q/p}$

2. LITERATURE REVIEW

In this section, we first introduce studies on the WP and its extensions, then we give a brief discussion on the VLSN search algorithms.

2.1. The Weber Problem and Its Extensions

In his pioneering work, Cooper [2] has addressed the MWP. The author noted that the objective function of the MWP is neither convex nor concave and hence it is NP-hard. A couple of exact solution procedures have been proposed for the MWP. In his early work, Rosing [9] has developed a Branch-and-Bound (BB) algorithm to solve the MWP, Krau [10] has utilized a column generation approach and a BB algorithm to tackle this problem. Then, Cooper [11] has developed the Alternate Location-Allocation (ALA) algorithm for obtaining approximate solutions of the MWP. Hansen et al. [12] have also focused on the MWP. They have considered all fixed points as potential facility sites, and then they have applied the ALA algorithm to determine proper locations for the facilities. Subsequently, two procedures which are based on *the furthest distance rule* and *the forbidden points rule* have been used within the ALA algorithm in order to generate efficient initial feasible solutions. Brimberg et al. [13] have proposed a solution approach for the MWP by exploiting a combination of Variable Neighbourhood Search (VNS) and the ALA algorithms. Gamal and Salhi [14] have developed a two-phase heuristic method which is named as a cellular heuristic. In another study, Salhi and Gamal [15] have proposed Genetic Algorithm (GA) to solve the MWP. Taillard [16] has suggested a decomposition heuristic to partition the MWP into smaller problems. Brandeau and Chiu [17] have addressed ALA heuristic and proposed statistical estimation approach for MWP. Aras et al. [18] have considered the MWP. The authors have concentrated on both rectilinear and Euclidean distance cases and they have proposed new quantization and Self-Organizing Map (SOM) algorithms by incorporating the properties of the distance function within their update rules. Liu et al. [19] have proposed Simulated Annealing (SA) to solve MWP for the rectilinear case.

To tackle MWP, two other studies have been accomplished by Lozano et al. [20] and Hsieh and Tien [21]. The methods in the last two works have been based on Kohonen's SOM algorithm.

The exact solution procedures of the CMWP were designed to enumerate all basic feasible solutions of the Transportation Problem (TP) polyhedron. Regarding the integrity of this approach, any feasible solution can be used as a start point in order to construct the connected graph of all basic feasible solutions. Sherali et al. [22] have introduced an exact solution method to tackle the rectilinear distance CMWP. In their proposed method, a reformulation of the problem as a mixed integer bi-objective linear programming model with Reformulation-Linearization Technique (RLT) is constructed. Al-Loughani [23] has also presented an exact method to deal with the Euclidean distance CMWP where a BB algorithm is designed. That algorithm has enumerated all vertices of the feasible region of the TP polyhedron. In addition, Sherali et al. [24] have developed a BB algorithm that is derived from a partitioning of the allocation space. The authors have addressed the CMWP for the Euclidean and l_p distances. Two bounding schemes have also been designed based on identifying a projected location space subproblem and a variant of the RLT by reformulating the CMWP as a Linear Programming (LP) relaxation problem. Another method developed by Sherali et al. [25] for the rectilinear distance CMWP is a convergent cutting plane algorithm. This approach is based on a Bilinear Programming (BP) problem whose decision variables are substituted by the difference of two non-negative variables. This approach has further been investigated by Sherali et al. [26]. The authors have used a distance proportional to the square of the Euclidean and the problem was transformed into a quadratic convex maximization problem. Sherali et al. [26] promoted an identical approach based on a BB algorithm to obtain strong upper bounds via a Lagrangean Relaxation (LR) scheme and a partitioning approach based on dichotomy that adopts a special structure of the transportation constraints.

In his early work, Cooper [2] has suggested the well-known Alternate Transportation-Location (ATL). Aras et al. [27] have devised a Mixed Integer Linear Programming (MILP) approximation of the CMWP, and then they have established three heuristic

methods to manage the constructed MILP problem with Euclidean, squared Euclidean and l_p distances. The experiments by Aras et al. [28] have indicated that earlier studies could be adopted and modified to solve the CMWP with rectilinear distance. In order to prove this, Aras et al. [29] have addressed the CMWP with rectilinear, Euclidean, squared Euclidean, and l_p distances and they have developed a SA algorithm, Threshold Accepting (TA) algorithm and GA. A perturbation-based heuristic method has been designed by Zainuddin and Salhi [30] which yields extraordinary results in comparison to the classical ATL when monitored on large-size instances given by Brimberg et al. [13]. A perturbation scheme was developed regarding the borderline customers to form clusters, and these customers were those that locate approximately half-way between their nearest and their second nearest facilities. Selected customer clusters were then organized and taken out temporarily during the construction of the TP. The clusters were introduced back again when finding the new location afterward. Recently, Luis et al. [31] have used the ATL algorithm by Cooper [2] with a Greedy Randomized Adaptive Search Procedure (GRASP) in order to obtain robust solutions.

Luis et al. [32] have also concentrated on the CMWP. The capacity restrictions are debated and it assumed that these capacities are given with a fixed number. They have inspired from Cooper's ALA heuristic where the facility locations are randomly defined at the first step of the algorithm. Instead of this, they have proposed a Region-Rejection based Approach (RRA) which forbids some areas that are not close to customers and which assure the initial location as possible as close to the density of the customers. They have developed four different extensions of the algorithm RRA with fixed radius (RRAFR), RRA with dynamic radius (RRADR), a cell-based approach with fixed radius (CBAFR) and cell-based-approach with dynamic radius (CBADR).

The MCMWP is a quite new problem comparing to the CMWP. To the best of our knowledge, Akyüz et al. [5] is the first work on the MCMWP. For other studies on the MCMWP, we refer to [6, 7, 8].

For all we know, the SSCMWP has first been addressed by Gong et al. [33]. The authors have suggested an iterative approach containing location and allocation phases known as Hybrid Evolutionary Method (HEM) to solve the SSCMWP. In the location phase, a GA is used to obtain the proper locations for the facilities. When the locations of facilities are established, the problem reduces to GAP. Following the previous steps, a LR scheme is employed to solve the resulting GAP in the allocation phase where the capacity constraints are dualized. Another work on the SSCMWP is accomplished by Manzour-al-Ajdad et al. [34]. They have proposed a heuristic to tackle SSCMWP. In their work, a SA is embedded in the improvement phase of ALA type heuristic. The results obtained by SA algorithm are more efficient than the results of the straightforward ALA type heuristic version.

As a special case of CMWP, we can consider discrete facility location problems. There is a vast amount of research interest on these problems. Some of them are as follows: Rönnqvist et al. [35] have suggested Single Source Capacitated Facility Location Problem (SSCFLP) with Euclidean distances that aims to locate the facilities on the candidate sites in order to minimize the total transportation cost under the capacity constraints and the single source assumption. The objective function tries to minimize the number of facilities as well as the distances between the facilities and customers. They have proposed a repeated matching algorithm for the SSCFLP. Pirkul et al. [36] have addressed the Multi-Commodity Capacitated Facility Location Problem (MCFLP). They have presented an MILP model of the problem and they have proposed an efficient heuristic based on a LR scheme, but instead of considering the whole of the plane as suggested by Cooper [2], they have also determined the locations of facilities between candidate places. For other studies, we refer to [37, 42, 43, 44, 45, 46, 47, 48, 49, 50]. On the other hand, for a more detailed literature survey on the continuous location-allocation problems we can cite the study by Brimberg et al. [47].

2.2. VLSN Search Algorithms and Their Applications

VLSN search algorithm is a Local Search (LS) algorithm with high efficiency; but very limited flexibility. Since the roots of our work consist of VLSN search algorithm, we

prefer to cite some of the crucial works on the VLSN search algorithms as follows: Yagiura et al. [48] have addressed the Multi-Resource Generalized Assignment Problem (MRGAP) which concerns assigning the tasks to the agents with respect to multi-resource constraints in order to minimize the total assignment cost where each task must be assigned to exactly one agent. To tackle the problem, the authors have proposed a TS with chained shift neighbourhood. Minic and Punnen [49] have also focused on the MRGAP. A VLSN search algorithm which combines a Variable Neighbourhood Search (VNS) procedure is constructed. Ahuja et al. [50] have proposed two VLSN search algorithms to tackle the Capacitated Minimum Spanning Tree (CMST) problem. Ergun and Orlin [51] have considered the symmetric Travelling Salesman Problem (TSP) for which they have proposed a VLSN search algorithm embedded with a Dynamic Programming (DP) approach.

Furthermore, Ahuja et al. [52] have addressed the Quadratic Assignment Problem (QAP) and have proposed a VLSN search algorithm which contains k-exchange neighbourhood structure to explore larger neighbourhoods. For a detailed survey on the VLSN search algorithms, we refer to Ahuja et al. [53]. Recently, Öncan et al. [54] have proposed VLSN search algorithms for partitioning type problems. Ahuja et al. [55] have addressed the Single-Source Capacitated Facility Location Problem (SSCFLP) which tackles with the optimal location of a number of facilities that will serve a set of customers with known locations. Similar to the SSCMWP, in the SSCFLP each customer must satisfy all its demand from exactly one facility, as well. Ahuja et al. [55] have claimed that SSCFLP is NP-hard and hence difficult to solve. This was their motivation to propose an efficient LS algorithm, namely a VLSN search algorithm, for the SSCFLP. Their VLSN search algorithm consists of customer exchange and facility move operations. These operations are realized on a specially constructed graph, i.e. the improvement graph. Their VLSN search heuristic has been tested on benchmark instances from the literature and on a real life instance, as well. Ahuja et al. [55] have reported that their VLSN search algorithm beats the LR heuristics proposed by Holmberg et al. [56] and Hindi and Pienkosz [57]. Furthermore, Ahuja et al. [55] have also stated that their VLSN search algorithm also outperforms CPLEX MILP solver in terms of both accuracy and efficiency.

3. DEFINITION OF THE SSCMWP

3.1. A Mathematical Programming Formulation for the SSCMWP

Let N be a set of customers whose known locations are indicated by $\mathrm{a}_j = (\mathrm{a}_{j1}, \mathrm{a}_{j2})^T$ and let q_j define the demand of a customer j for $j = 1,...,N$. Furthermore, let M denote a set of facilities whose unknown locations are indicated by $\mathrm{x}_i = (\mathrm{x}_{i1}, \mathrm{x}_{i2})^T$ and let s_i stand for the supply quantity of facility i for $i = 1,...,M$. We define binary decision variable w_{ij} which is equal to 1 if and only if the demand of customer j is completely satisfied from facility i for all $i = 1,...,M$ and $j = 1,...,N$. As it can be seen, this formulation consists of two decision variables: x_i and w_{ij}. Moreover, c_{ij} denotes unit shipment cost per unit distance. The distances are computed using the rectilinear distance $d(\mathrm{x}_i, \mathrm{a}_j) = |\mathrm{x}_{i1} - \mathrm{a}_{j1}| + |\mathrm{x}_{j2} - \mathrm{a}_{j2}|$ between facility i and customer j. Finally, the objective function consists of the minimization of total transportation cost with respect to capacity constraints and under a single source assumption. Then, a mathematical programming formulation of the SSCMWP can be given as follows:

$$\text{SSCMWP}: \quad \min z = \sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij} q_j c_{ij} d(\mathrm{x}_i, \mathrm{a}_j) \tag{3.1}$$

subject to

$$\sum_{j=1}^{N} q_j w_{ij} \le s_i \qquad i = 1,...,M, \tag{3.2}$$

$$\sum_{i=1}^{M} w_{ij} = 1 \qquad j = 1,...,N, \tag{3.3}$$

$$w_{ij} \in \{0;1\} \qquad i = 1,...,M; j = 1,...,N. \tag{3.4}$$

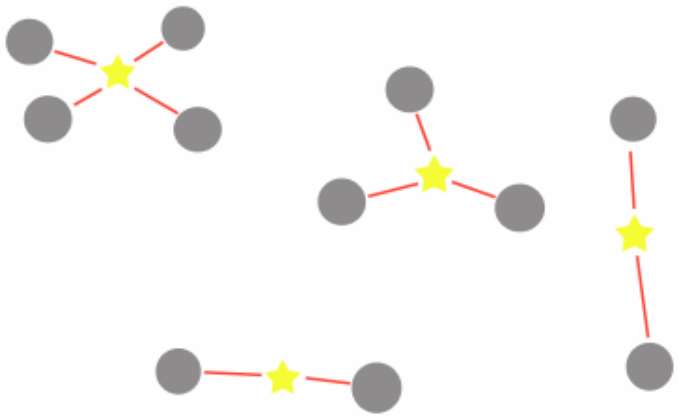

Figure 3.1. Illustration of an SSCMWP Instance

In Figure 3.1. we illustrate a feasible solution of SSCMWP instance with 11 customers and 4 facilities. The customers are indicated by circles and the facilities are shown by stars. Observe that each customer is assigned to exactly one facility.

The objective function (3.1) calculates the total transportation cost which is assumed to be directly proportional to the customer demands times the distance between facility i and customer j which is denoted by the rectilinear distance. Constraints (3.3) guarantee that every customer must be served from exactly one facility while a facility is allowed to serve more than one customer. Constraints (3.2) ensure that the capacity of each facility is not exceeded. Finally, we should note that the SSCMWP has the balancedness assumption. That is to say, the total demand of all the customers is exactly equal to the total capacity of the facilities. In other words, $\sum_{j=1}^{N} q_j = \sum_{i=1}^{M} s_i$ holds.

3.2. The Single Source Transportation Problem

The SSTP can be considered as a special case of the SSCMWP. Note that the constraints of both the SSCMWP and the SSTP are the same but their objective functions are different. An MILP formulation of the SSTP is given below.

$$\text{SSTP}: \quad \min z = \sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}\overline{C}_{ij} \tag{3.5}$$

subject to

$$\sum_{j=1}^{N} q_j w_{ij} \le s_i \qquad i = 1,...,M\ , \tag{3.6}$$

$$\sum_{i=1}^{M} w_{ij} = 1 \qquad j = 1,...,N\ , \tag{3.7}$$

$$w_{ij} \in \{0;1\} \qquad i = 1,...,M\ ;\ j = 1,...,N\ . \tag{3.8}$$

Here, $\overline{C}_{ij}$ indicates the transportation cost of sending one unit from facility i to customer j. Note that whenever the locations of facilities are known, the SSCMWP reduces to the SSTP. In other words, for $\overline{C}_{ij} = q_j.c_{ij}.d\left(\mathrm{x}_i, \mathrm{a}_j\right)$, the SSTP becomes the SSCMWP. Furthermore, when constraints (3.6) are of type "≥" instead of "≤", then the SSTP refers to the CAP [58].

4. SOLUTION PROCEDURES

First of all, we present the Discrete Approximation Problem (DAP) heuristic for the SSCMWP. Then, we present ALA type heuristic for the SSCMWP.

4.1. Discrete Approximation Problem Heuristic for the SSCMWP

On the rectilinear distance case, Thisse et al. [59] have demonstrated that the optimum facility locations lie on the intersection points obtained by drawing the extreme directions of the block norm on customer locations for the WP by generalizing Wendell and Hurter's [60] results. As for instance, we can state that the optimum solution of the rectilinear distance WP lies in one of the points on the intersection of vertical and horizontal lines drawn by passing through customer locations.

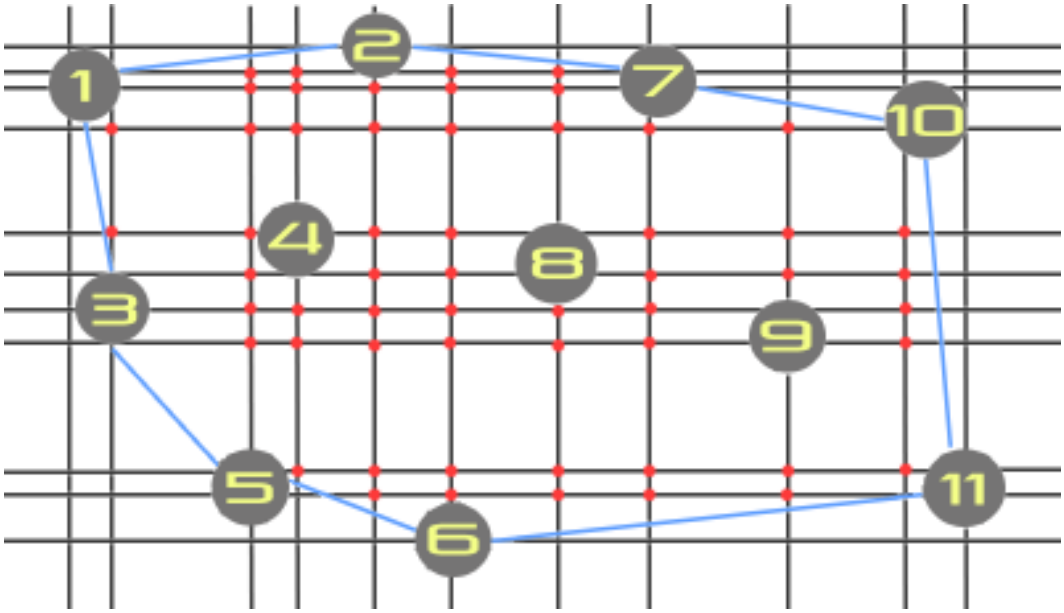

Figure 4.1. Illustration of Candidate Facility Locations in the DAP formulation

In Figure 4.1., there are 11 customers. The candidate facility locations lie within the convex hull defined by customers 1, 2, 7, 10, 11, 6, 5 and 3. All intersection points obtained by drawing vertical and horizontal lines through customer locations and remaining within the convex hull are candidate facility locations. Notice that these vertical and horizontal lines constitute the extreme rays of the rectilinear-norm which is a block norm.

These properties are also valid for the multi-facility extensions of the WP, namely, MWP, CMWP and SSCMWP, because these problems reduce to M WPs when feasible allocations are given. Furthermore, Hansen et al. [61] have generalized the properties obtained for the rectilinear distance WP by Wendell and Hurter [60] to show that the optimal solutions of the two-dimensional location problems with rectilinear distance function always occur within the convex hull of the customer locations and at the intersection of vertical and horizontal lines drawn through them. Aras et al. [28] have taken advantage of these properties. They have reformulated the rectilinear distance CMWP as an MILP problem by restricting optimal facility locations to belong within the candidate location set.

These results can also be applied for the rectilinear distance SSCMWP. When there are a finite number of candidate points at which facilities can be located, the optimal solution of the corresponding rectilinear distance SSCMWP can be obtained by solving an MILP problem namely the Discrete Approximation Problem (DAP). Given P a set of candidate points denoted by p=1,...,P, we define a binary decision variable $w_{ij}^{\ p}$ which is equal to 1 if and only if the demand of customer j is completely satisfied from facility i that is located at candidate point p. Note that whenever the candidate points p=1,...,P are selected as the intersection points obtained by vertical and horizontal lines drawn through customers, the solution of the DAP becomes the optimum solution of the rectilinear distance SSCMWP, according to the results by Wendell and Hurter [60]. Then, the DAP formulation is presented as follows :

$$\text{DAP}: \quad \min z = \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{p=1}^{P} c_{ij}{}^{p} q_j w_{ij}{}^{p} \tag{4.1}$$

subject to

$$\sum_{j=1}^{N} q_j w_{ij}{}^{p} \le s_i y_i{}^{p} \qquad i = 1,...,M \ , \ p = 1,...,P, \tag{4.2}$$

$$\sum_{p=1}^{P}\sum_{i=1}^{M} w_{ij}{}^{p} = 1 \qquad j = 1,...,N, \tag{4.3}$$

$$\sum_{p=1}^{P} y_i{}^{p} = 1 \qquad i = 1,...,M, \tag{4.4}$$

$$y_i{}^{p} \in \{0;1\} \qquad i = 1,...,M \ , \ \ p = 1,...,P, \tag{4.5}$$

$$w_{ij}{}^{p} \in \{0;1\} \qquad i = 1,...,M \ , \ j = 1,...,N \ , \ p = 1,...,P \ . \tag{4.6}$$

Here, binary decision variable $y_i{}^{p}$ is equal to 1 if and only if facility i is located at candidate point p. Although solving DAP yields the optimal solution of the rectilinear distance SSCMWP, this task may require drastic CPU time. As for example, consider a SSCMWP instance with 5 facilities and 100 customers. For this case, the DAP formulation includes $5 \times 100 \times 100^2 + 5 \times 100^2$ binary decision variables and $5 \times 100^2 + 100 + 5$ constraints. Hence, we can say that for large instances solving DAP to optimality is not a viable option.

On the other hand, note that solving the DAP with a CPU time limit may yield an upper bound for the SSCMWP. Therefore, we employ this approach as a heuristic and we name it as “DAP heuristic”.

4.2. ALA Type Heuristic

Location-Allocation Problems (LAPs) try to find the optimal locations of a set of facilities and optimal allocations of customer demands to the facilities with respect to capacities of facilities and demand restrictions at minimum total transportation cost. Any LAP becomes a pure multi-facility location problem when an allocation scheme is known. Conversely, it becomes a pure allocation problem when facility locations are defined. First of all, we suggest an ALA type heuristic for the rectilinear distance SSCMWP which is inspired from the seminal work by Cooper [2].

Our ALA type heuristic consists of two phases which are performed consecutively and iteratively. In the first phase, we aim to locate the facilities in the continuous plane and in the second one, our goal is to find the optimal allocation plan of customers to facilities by solving a TP. These two phases are performed repeatedly until there is no improvement in the objection function value. We know that the quality of the final solution that ALA computes depends on its initial solution very much where the facility locations are randomly determined at the initialization step of the ALA type heuristic.

In our implementation of the ALA type heuristic for the SSCMWP, we first try to locate the facilities and then assign each customer to a single facility with considering capacity constraints. We change the allocations by solving an MILP problem (i.e. SSTP) then, we determine new facility locations. Namely, in the location phase, we have to solve M WPs by running M times Weiszfeld’s Algorithm in order to find the optimum facility locations for each facility. The WP is formulated as follows:

$$\min_{x_i} z = \sum_{j=1}^{N} \overline{C}_{ij} d(\mathrm{x}_i, \mathrm{a}_j) \qquad i = 1,...,M \tag{4.7}$$

where $\overline{C}_{ij}$ is computed as $\overline{C}_{ij} = q_j c_{ij} \overline{w}_{ij}$ holds. Here $\overline{w}_{ij}$ stands for the fixed assignment of facility i to customer j. Recall that the WP can be solved by running the Weiszfeld's algorithm [62] or one of its generalizations [63]. Next, we give the steps of the Weiszfeld's algorithm.

4.2.1. Weiszfeld's Algorithm for the SSCMWP

Weiszfeld's algorithm is an iterative method which aims to optimally locate a single facility that serves its customers in the continuous plane. Namely, the objective is to find the optimal location in order to minimize total transportation cost.

Given the assignments of customer j to facility i, i.e. $\overline{w}_{ij}$, the steps of the Weiszfeld's algorithm is as follows :

Step 1: $\chi \leftarrow 0$

Step 2: Initialize the facility location.

$$\mathrm{x}_{i1}^{0} = \frac{\sum_{j=1}^{N} \overline{w}_{ij} q_j \mathrm{a}_{j1}}{\sum_{j=1}^{N} q_j \overline{w}_{ij}} \text{ and } \mathrm{x}_{i2}^{0} = \frac{\sum_{j=1}^{N} \overline{w}_{ij} q_j \mathrm{a}_{j2}}{\sum_{j=1}^{N} q_j \overline{w}_{ij}} \quad ; \tag{4.8}$$

Step 3: Calculate the rectilinear distances $d_{ij}^{(\chi)} = d\left(\mathrm{x}_i^{(\chi)}, \mathrm{a}_j\right)$ between new locations of facilities and the locations of customers.

Step 4: Update the location of facility i for the next iteration :

$$x_{i1}^{(\chi+1)} = \frac{\sum_{j=1}^{N} \frac{\overline{w}_{ij} q_j \mathrm{a}_{i1}}{d_{ij}^{(\chi)}}}{\sum_{j=1}^{N} \frac{\overline{w}_{ij} q_j}{d_{ij}^{(\chi)}}} \text{ and } x_{i2}^{(\chi+1)} = \frac{\sum_{j=1}^{n} \frac{\overline{w}_{ij} q_j \mathrm{a}_{i2}}{d_{ij}^{(\chi)}}}{\sum_{j=1}^{N} \frac{\overline{w}_{ij} q_j}{d_{ij}^{(\chi)}}}; \tag{4.9}$$

Step 5: Increase iteration counter $\chi \leftarrow \chi + 1$

Step 6: Go to Step 3 until the termination criteria is satisfied.

In this work, we have defined two stopping criteria for the Weiszfeld's Algorithm. The first one is an iteration limit number that is equal to 5000. The second one consists of the changes in the facility locations. In other words, when both abscise and ordinate are changed smaller than 0.0001, then the algorithm stops.

4.2.2. The Outline of the ALA Type Heuristic

The steps of ALA Type Heuristic are described as follows :

Step 1: Locate M facilities at arbitrarily selected points $\mathrm{x}_i = (\mathrm{x}_{i1}, \mathrm{x}_{i2})^T$ for $i = 1,...,M$ within the convex hull of customers.

Step 2: For each facility i and customer j, calculate the rectilinear distance $d(\mathrm{x}_i, \mathrm{a}_j)$ between them and set the new transportation cost as $\overline{C}_{ij} = q_j c_{ij} d(\mathrm{x}_i, \mathrm{a}_j)$.

Step 3: Determiner feasible allocations w_{ij}^* by solving the SSTP.

Step 4: Solve M WPs to relocate M facilities.

Step 5: Repeat Steps 4-3 until either facility locations $\mathrm{x}_i = (\mathrm{x}_{i1}, \mathrm{x}_{i2})^T$ for $i = 1,...,M$ or allocations w_{ij}^* for $i = 1,...,M$; $j = 1,...,N$ remain unchanged.

ALA type heuristic is a simple and quite efficient heuristic; however the quality of its final solution depends very much on its initial solution. The most difficult part of the proposed ALA type heuristic is Step 3 which requires solving the SSTP, namely a MILP problem. We have employed CPLEX MILP solver to deal with the SSTP. However, for large size instances, using CPLEX MILP solver may require excessive CPU times and hence for these instances (especially instances bigger than 10 facilities 100 customers), ALA type heuristic was unable to obtain feasible solutions in acceptable computation times.

4.3. ALA with VLSN Search Algorithm

For large instances, the CPU time required for solving the SSTP via CPLEX MILP solver becomes excessive. One way to deal with this difficulty is to solve the SSTP by running a LS algorithm. For that purpose, we have employed the VLSN search algorithm which uses the matching neighbourhood. This neighbourhood is proposed by Öncan et al. [54] for partitioning type problems including the SSTP.

A LS algorithm tries to find better feasible solutions starting from an initial feasible solution, by replacing new feasible solutions with the previous ones repeatedly until some stopping criteria is satisfied. A crucial issue in the design of LS algorithms is the choice of a proper neighbourhood structure. In the next discussion, we introduce the matching neighbourhood which is employed within our VLSN search algorithm.

4.3.1. The Matching Neighbourhood for the SSTP

Let $N = \{1,2,...,J\}$ be a finite set and $\{F_1, F_2, ..., F_I\}$ be families of prescribed subsets of N. Let $A = (A_1, A_2, ..., A_I)$ be an ordered partition of N, i.e. $\cup_{i=1}^{I} A_i = N$ and $A_i \cap A_k = \varnothing$ for $i \neq k, i, k = 1,...,I$. We say that the ordered partition A is *feasible* if

and only if $A_i \in F_i$ for all i. For the SSTP case, F_i is defined as $\{A_i : A_i \subseteq N; \sum_{j\in S} q_j \le s_i\}$ for $i = 1,...,I$. For each $i = 1,2,...,I$ and $S \in F_i$ a cost function $c_i(S)$ is also prescribed where $c_i(S) = \sum_{j\in S} c_{ij}$ for $i = 1,...,I$. Then the *partitioning problem* is to find a feasible partition A such that $f(A) = \sum_{i=1}^{I} c_i(A_i)$ is minimum. Furthermore, for any ordered partition $A = (A_1, A_2,..., A_I)$ define $w_{ij} = 1$ if $j \in A_i$, $i = 1,2,...,I$ and zero otherwise. Then it can be verified that this solution is feasible for the SSTP if and only if A is a feasible partition.

For each set A_i in A, choose a subset B_i. Note that B_i could be the empty set. The ordered collection $B = (B_1, B_2,..., B_I)$ is called an ejection vector of A. Given A and an ejection vector B, we construct a bipartite graph $\widetilde{G}$, called the improvement graph as follows. The generic bipartition of its vertex set is $\widetilde{V}_1 \cup \widetilde{V}_2$ where $\widetilde{V}_1 = \{t_1, t_2,..., t_I\}$ and $\widetilde{V}_2 = \{r_1, r_2,..., r_I\}$. Node t_i in $\widetilde{V}_1$ represents the subset A_i and node r_j in $\widetilde{V}_2$ indicates the subset B_j. Furthermore, an edge (t_i, r_j) is added to the graph $\widetilde{G}$ if and only if $(A_i \cup B_j) \backslash B_i \in F_i$ holds. For example, in the case of SSTP verifying this condition is equivalent to testing $q(A_i) - q(B_i) + q(B_j) \le s_i$ that is node t_i in $\widetilde{V}_1$ is connected to a node r_j in $\widetilde{V}_2$ by an edge (t_i, r_j) if and only if $q(A_i) - q(B_i) + q(B_j) \le s_i$ holds. Here, $q(A_i)$ and $q(B_i)$ denote the total customer demands in sets A_i and B_i respectively. The edge, (t_i, r_j) signifies adding B_j to A_i while B_j is ejected from A_i. The cost of (t_i, r_j) is given by $\varsigma_{ij} = c_i(B_j) - c_i(B_i)$ where, for the SSTP case $c_i(B_i) = \sum_{j\in B_i} c_{ij} q_j d(x_i, a_j)$ holds. Note that for each $i = 1,2,...,I$ the edge (t_i, r_i) is in $\widetilde{G}$ and $\varsigma_{ij} = 0$. Let $M = \{(t_i, r_j) | i = 1,2,...,I; j = 1,2,...,I\}$ be any perfect matching in $\widetilde{G}$. Note that $\widetilde{G}$ contains at least one perfect matching. Then $\overline{A} = (\overline{A}_1, \overline{A}_2,..., \overline{A}_I)$ is a feasible ordered partition. The operation of building $\overline{A}$ from A is called an *M-matching exchange*. The simple matching neighbourhood of S with respect to a given ejection vector $B = \{B_1, B_2,..., B_I\}$, denoted by $M_I(B)$, is the collection of all feasible

ordered partitions each of which can be obtained by a M-matching exchange from A using some perfect matching M of $\widetilde{G}$. As shown by Öncan et al. [54] computing the best member in $M(A)$ is NP-hard for the SSTP therefore we resort to a heuristic approach to find the ejection vector $B=\{B_1,B_2,...,B_I\}$. For that purpose we have employed a LS scheme. Given a feasible partition A, we assume that a heuristic procedure $H(A)$ is available which with input A searches approximately $M(A)$ for an improving solution and outputs such a solution if obtained. The algorithm is terminated when $H(A)$ fails to detect an improving solution. The task of $H(A)$ is to search for an improving solution in $M(A)$. However, we already know that finding an efficient search scheme seems difficult. Note that a best solution in $M(A)$ is a best solution in $M_I(A)$ for some ejection vector B. Further, $M_I(A)$ can be searched for a best improving solution in polynomial time. Thus, in case we can generate a good enough ejection vector B, we are likely to find an improving solution in $M(A)$ if one exists. It is possible to consider a simple rule to generate the ejection vectors. For that purpose, we use two parameters λ and γ which stand for the minimum and maximum number of items to be ejected, respectively. The components of the ejection vector B have cardinality between these parameters. It is possible to set $\lambda=0$ which implies $B_r=\varnothing$ for some component B_r of B and in all our experiments we set $\lambda=0$ or 1. On the other hand, λ is set to a random number between 1 and $|A_r|$, for each component A_r of A. As a straightforward ejection rule we have applied the random ejection rule because of its simplicity. Note that it is possible to employ many different ejection rules [54]. However for the SSTP arising in the allocation phase of the ALA type heuristic, we have observed that the random ejection yields acceptable solutions within reasonable computation times. The random ejection rule is as follows. For a given partition $A=\{A_1,A_2,...,A_I\}$ eject a random subset B_r from each non-empty A_r for $r=1,...,I$ such that $|B_r|=k_r$ where k_r is a random number between λ and $\min\{|A_r|,\gamma\}$. We set $B_r=\varnothing$ when $A_r=\varnothing$. Yet another ejection rule is the Recency Memory Based (RBM) ejection rule. We keep track of ejected customers and the ejection probability of each customer j is inversely proportional to the number of times customer j is ejected from

facility i. The VLSN search algorithm as presented above is called as Allocation Improvement VLSN (AI-VLSN) search algorithm.

The proposed VLSN search heuristic can also be used to construct a feasible solution. To accomplish this, given an infeasible solution we relax constraints (3.2) of the SSTP formulation and we add a penalty for the violating constraints. In other words, we construct an improvement graph by defining the cost $\tau_{ij} = \max\{0, (q(A_i / B_i) + q(B_j) - s_i)\}$ for each edge (t_i, r_j). The modified graph is called as feasibility improvement graph on which we perform M-matching exchange operations using the random ejection rules until a feasible solution is reached. The VLSN search algorithm with the feasibility improvement scheme is named as Feasibility Chasing VLSN (FC-VLSN) search algorithm.

4.3.2. The Outline of the ALA with VLSN Search Algorithm

Step 1: Initial assignment of customers to facilities is performed randomly.
Step 2: A feasible assignment is constructed by running FC-VLSN search algorithm.
Step 3: The facility locations are determined by running Weiszfeld's Algorithm and the distances between customers and facilities are computed.
Step 4: Given the feasible solution obtained in Step 2, an improved feasible solution for the SSCMWP is reached by running AI-VLSN search heuristic algorithm.
Step 5: Step 4 and Step 5 are repeated until there is no improvement in the objective function value of the SSCMWP.
Step 6: Steps 1-5 are repeated until a fixed number of SSCMWP solutions are generated and the best SSCMWP solution among them is output.

4.4. ALA with Tabu Search Algorithm

TS is an iterative LS method applied for several combinatorial optimization problems. Starting from an initial solution, the TS method moves at each iteration from the current solution to the best one in a subset of its neighbourhood, even if this causes a deterioration in the objective function value. To avoid cycling, solutions possessing

some attributes of recently visited solutions are declared *forbidden* or *tabu* for a given number of iterations, called *tabu tenure*. The algorithm stops whenever a preset criterion is satisfied. During the search, the algorithm maintains short term and long term memory structures. When an attribute is declared tabu, all solutions possessing this attribute are implicitly declared tabu. However, in fact some of these solutions may never have been considered by the search. To remedy this, an aspiration criterion is defined to override the tabu status of a solution. One common aspiration criterion is to allow tabu solutions yielding better solution values than that of the best known solution. In our TS, we apply an extension of the aspiration level concept by associating an attribute to each assignment of customer j to facility i. Hence, the attribute set $\Omega(\pi)$ of solution π consists of customer-facility assignments. We define the tabu rule as follows. If customer i is removed from the facility j, reinserting it into that facility is forbidden for the next θ iterations, where θ is the tabu tenure. The last iteration for which attribute (i, j) is declared tabu is denoted by $\beta_{(i,j)}$. The tabu status of an attribute can be revoked if it leads to a solution with smaller cost than that of the best solution identified having that attribute. The aspiration level $\gamma_{(i,j)}$ of an attribute is initially set equal to the cost of initial solution π_0 where customer j is assigned and to ∞ otherwise. At every iteration, the aspiration level of each attribute $(i, j) \in \Omega(\pi)$ of the current solution is updated to $\min\{\gamma_{(i,j)}, c(\pi)\}$, where $c(\pi)$ stands for the cost value of solution π. Given a solution π, its neighbour solutions are defined by $N(s)$. At each iteration, we consider a subset $T(\pi) \subseteq N(\pi)$ which consists of all solutions $\overline{\pi} \in N(\pi)$ reachable from π without incurring the risk of cycling, i.e., that are not tabu or that satisfy the aspiration criterion.

4.4.1. An Attribute Based Tabu Search Algorithm for the SSTP

We now describe the steps of the TS algorithm used to solve the SSTP arising in the allocation phase of the ALA type heuristic for the SSCMWP.

In order to construct an initial feasible solution we randomly locate facilities then, we perform FC-VLSN search algorithm. Then, this initial solution is set as the best solution π^*. We first initialize the number of times attribute (i, j) has been added to the solution. This parameter is set equal to 1 for all assignments (i, j) in the current solution (i.e. $\kappa_{(i,j)} = 1$ for all $(i, j) \in \Omega(\pi)$, and $\kappa_{(i,j)} = 0$ for all $(i, j) \notin \Omega(\pi)$). Then, the last iteration for which attribute (i, j) is declared as tabu is set equal to 0 for all assignments (i.e., $\beta_{(i,j)} = 0$ for all (i, j)). The aspiration levels $\gamma_{(i,j)}$ are set equal to the initial solution value $c(\pi)$ for all assignments in the current solution (i.e., $\gamma_{(i,j)} = c(\pi)$ for all $(i, j) \in \Omega(\pi)$ and $\gamma_{(i,j)} = \infty$ for all $(i, j) \notin \Omega(\pi)$). The number of times customer j has been considered for the neighbourhood search is set equal to 0 for all customers (i.e., $\delta_j = 0$ for all $j = 1,...,J$). Finally, improvement iteration counter χ is set equal to 0.

The first step of our neighbourhood scheme is the customer selection procedure which is crucial for the performance of the neighbourhood $N(\pi)$ and hence to the performance of the TS algorithm. As a simple rule, we can randomly select customers. However, during the run of the TS algorithm this may cause the algorithm to repeatedly visit the same customers and hence the same space without improving the solution. Especially, when the TS algorithm runs for a small number of iterations, some customers are selected less often than the others. Therefore, we propose a diversification scheme for the customer selection considering the customer visit frequencies, i.e. δ_j which denotes the number of times a customer j has been selected for the *exchange* or *swap* operation. The customer selection is performed with probabilities proportional to the inverse of the δ_j values. Hence, the lower the value of δ_j, the higher the probability of selecting customer j. When all customers have been visited an equal number of times, we apply the random selection rule for at most Λ non-improving iterations. If the random selection of customers has not yielded an improved solution for successive Λ iterations, we again switch to customer selection by visit frequencies rule, and so on.

Suppose that customer j' which is assigned to facility i', is selected for an exchange operation. Then among all customers which are not assigned to facility i' and the most profitable one, say i'' is considered for an exchange operation. In other words, the assignments (i', j') and (i'', j'') are replaced with (i', j'') and (i'', j') respectively. This yields an exchange operation. By performing this exchange operation, we move from solution π to solution $\bar{\pi}$ then the profit of this operation is computed as follows:

$$\vartheta_{(\pi)}^{(\bar{\pi})} = \left(\varphi_{i'j'} q_{j'} + \varphi_{i''j''} q_{j''}\right) - \left(\varphi_{i''j'} q_{j'} + \varphi_{i'j''} q_{j''}\right). \tag{4.10}$$

Note that performing the most profitable exchange operation can yield an infeasible solution. Therefore, the LS phase has to be confined with feasible moves. In case it is not possible to find a feasible and profitable move for a predetermined number of iterations then the TS algorithm may stop. Therefore, assume that by an exchange operation we move from a solution π to a solution $\bar{\pi}$ then it means that we incur a profit of $\vartheta_{(\pi)}^{(\bar{\pi})} - \zeta(\bar{\pi})$, where

$$\zeta(\bar{\pi}) = \rho \max\left\{0, \left(\sum_{i=1}^{M}\left(\sum_{j=1}^{N} q_j \hat{w}_{ij} - s_i\right)\right)\varphi_{ij}\right\}. \tag{4.11}$$

In case, the new solution π is infeasible and hence $\zeta(\bar{\pi})$ is positive then we perform the VLSN feasibility chasing procedure to escape form infeasibility. In other words, the cost of the new solution $\bar{\pi}$ is computed as

$$c(\bar{\pi}) = c(\pi) - \vartheta_{(\pi)}^{(\bar{\pi})} + \zeta(\bar{\pi}). \tag{4.12}$$

The next step after the customer selection is to evaluate a subset $T(\pi)$ of neighbour solutions. For this purpose, we first initialize $T(\pi)=\varnothing$. Then, for each neighbour solution $\bar{\pi}$ of the current solution π, if there exists an attribute $(i,j)\in\Omega(\bar{\pi})\backslash\Omega(\pi)$ such that $\beta_{(i,j)}<\chi$ or $c(\bar{\pi})<\gamma_{(i,j)}$, we set $T(\pi)=T(\pi)\cup\{\bar{\pi}\}$. In other words, we add to $T(\pi)$ the neighbour solutions $\bar{\pi}$ such that either $\bar{\pi}$ is non-tabu or the cost of $\bar{\pi}$ is lower than the aspiration level $\gamma_{(i,j)}$ of one of the attributes $(i,j)\in\Omega(\bar{\pi})$. To penalize the vertices frequently added into the solution, we define a penalized cost value for each solution $\bar{\pi}$ in the subset of neighbourhoods $T(\pi)$ as follows:

$$p(\bar{\pi})=\begin{cases} c(\bar{\pi})+\phi\, c(\pi)\sqrt{IJ}\sum\limits_{(i,j)\in A(\bar{\pi})/\,A(\pi)}\kappa_{ij}/\chi, & \text{if } c(\bar{\pi})\geq c(\pi) \\ c(\bar{\pi}), & \text{otherwise} \end{cases} \quad . \tag{4.13}$$

Here, ϕ is a factor used to adjust the intensity of the diversification, while the square root factor $\sqrt{IJ}$, which was first proposed by Taillard [64] in the context of the Vehicle Routing Problem (VRP), is used to compensate for instance size. Finally, considering all solutions $\bar{\pi}$ in $T(\pi)$ we identify the solution π' with minimum penalized cost value $p(\pi')$.

Considering the new solution s' we update the following parameters. First, we update the tabu status of the removal of customer j from facility i, $(i,j)\in\Omega(\pi)\backslash\Omega(\pi')$ in order to forbid the pair (i,j) from entering into the solution for at least θ iterations (i.e., $\beta_{ij}=\chi+\theta$). Second, for each attribute $i\in\Omega(\pi')\backslash\ \Omega(\pi)$, the number of times attribute (i,j) has been added to the solution is increased by one (i.e., $\kappa_{(i,j)}=\kappa_{(i,j)}+1$). Third, if the new solution is better than the current best solution (i.e., $c(\pi')<c(\pi^*)$) we

update the current best solution (i.e., $\pi^* = \pi'$). Finally, for each vertex of the new solution we update its aspiration level (i.e., $\gamma_{ij} = \min\{\gamma_{ij}, c(\pi')\}$ for all $(i, j) \in \Omega(\pi')$).

Several stopping criteria can be adopted for the TS algorithm such as a time limit, an iteration limit or an absence of improvement for a given number of iterations. We choose our criterion so as to achieve a balance between computation time and solution quality. The TS algorithm stops when the iteration counter χ reaches the iteration limit η.

4.4.2. The Outline of the ALA with Tabu Search Algorithm

Step 1: Initial assignment of customers to facilities is done randomly.
Step 2: A feasible solution is constructed by FC-VLSN search algorithm.
Step 3: The locations of facilities are found by running the Weiszfeld's algorithm and the distances with the customers and facilities are calculated.
Step 4: The objective function value of the SSCMWP is computed.
Step 5: TS phase is processed in order to reach an improved SSTP solution**.**
Step 6: Steps 2-5 are repeated until a specified number of new solutions are generated and the best solution is output.

5. COMPUTATIONAL RESULTS

In this section, we present computational results obtained with our solutions procedures implemented on the SSCMWP. The experiments are performed on a Dell Server PE2900 with two 3.0 GHz Core2Duo Processors and 4 GB RAM operating within Microsoft Windows Server 2003 environment in C++. Cplex 11.0 with default options is used as a subroutine to solve the resulting MILPs which are part of the heuristic algorithms. We have totally considered 35 test instances. Benchmark test instances for the CMWP from OR-library are used to assess the performance of the proposed algorithms developed for the rectilinear distance SSCMWP. These instances can be grouped in two classes. The first class consists of 25 instances with size ranging as $4 \le M \le 10$ and $8 \le N \le 100$ and the second class involves 10 instances with size ranging as $10 \le M \le 50$ and $50 \le N \le 250$. The instances in the first class can be considered as small instances and the ones in the second class can be classified as large instances.

In our experiments, we have especially focused on small instances because we can not obtain solutions of large instances using the DAP heuristic and the ALA type heuristic in reasonable computation times. Recall that the DAP heuristic may yield the optimal solution for the SSCMWP at the expense of excessive CPU time. In our experiments, we allow DAP heuristic to run up to 7200 seconds. Thus, notice that for the instances, for which CPLEX MILP solver has run for less than 7200 seconds, the optimum solution is reached. In Table 5.1, we give the results obtained with DAP heuristic and ALA type heuristic on small instances. In Table 5.1., the first column includes the names of the test instances. The second and the third columns in the Table 5.1 present the results obtained with the DAP heuristic. CPLEX MILP solver is run with a CPU time limit of 7200 seconds. Therefore, for most of the cases UB indicates the best upper bound found by CPLEX MILP solver within time limit of 7200 seconds. The last two columns in Table 5.1., the results computed by ALA-CPX are given. As it can be

observed for the second and third columns of Table 5.1, we could only obtain optimal solutions for first two instances.

We propose three versions of the ALA-type heuristic: ALA-CPX, ALA-VLSN and ALA-TS. These three versions are different in terms of their solution approach employed to solve the SSTP which arises in the allocation phase of the ALA-type heuristic (i.e. Step 3). In the first version of ALA-type heuristic (i.e. ALA-CPX), the SSTP is solved to optimality by using CPLEX MILP solver. This is straightforward implementation of the ALA-type heuristic for the SSCMWP. In the second version of the ALA type heuristic (i.e. ALA-VLSN), the SSTP is solved by the VLSN search heuristic presented in Section 4.3. In the third version (i.e. ALA-TS), the SSTP is solved by TS algorithm presented in Section 4.4. We have also considered the DAP heuristic for the SSCMWP where the DAP formulation is solved by CPLEX MILP solver. As a result, we report computational results with four heuristics. These are DAP, ALA-CPX, ALA-VLSN and ALA-TS. Furthermore, two versions of the ALA-VLSN are designed. These are ALA-VLSN-R and ALA-VLSN-RBM. ALA-VLSN-R stands for the ALA-VLSN search algorithm with random ejection rule. This ejection mechanism is performed up to $I \times J$ times and the best feasible solution is reported. ALA-VLSN-RBM denotes the ALA-VLSN search algorithm with recency based Ejection rule. On the other hand, we have two versions of ALA-TS. These are TS-with feasible chasing phase (ALA-TSwFC) and TS-without feasible chasing phase (ALA-TSwoFC).

Table 5.2 and Table 5.3 contain the computational results including both CPU time and upper bounds obtained with ALA-VLSN and ALA-TS respectively. When we consider large instances, none of the DAP and ALA-CPX were able to yield upper bounds in reasonable CPU times (i.e. with a running time limit of 2 hours). Hence, in Table 5.3, we report only the results obtained with ALA-VLSN search algorithm and ALA-TS heuristic. The first column indicates the names of the test instances, columns 2, 4, 6 and 8 indicate the CPU times (in seconds) required and columns 3, 5, 7 and 9 are for upper bounds obtained.

In Table 5.4, a summary of the computational results are reported. In this table, the average CPU times and average upper bound values of the proposed algorithms on small instances are given. We can observe that the best upper bounds are obtained by ALA-VLSN-RBM. On the other hand, ALA-VLSN-R yields also promising results in terms of efficiency. Generally speaking, we can conclude that ALA-VLSN algorithms yield more promising results than the ALA-TS algorithms do. The accuracies of the proposed algorithms are illustrated with Figure 5.3.

In comparison between ALA-TS and ALA-VLSN, it can be easily noticed that ALA-VLSN is more efficient than ALA-TS in average. In our TS algorithm, only swap moves are performed while in ALA-VLSN multiple swap operations are allowed through the matching neighbourhood. Therefore, we expect ALA-VLSN to be more appropriate in reaching very good solutions in reasonable computation times.

Finally, in Table 5.5., we report computational experiments with the ALA-TS-woFC for different stopping criterion. In Table 5.5., we consider maximum iteration numbers 200, 500, 1000, 2000 and 3000, and we take non-improving iteration numbers of 50, 125, 250, 500 and 750. For a set of combinations of maximum iteration number and number of non-improving iterations, we have performed some experiments on small instances. We observed that when the CPU time increases, the accuracy of the ALA-TS-woFC improves considerably. We should note that in our experiments for ALA-VLSN and ALA-TS, the maximum iteration number is set to 100 and the non-improving iteration limit is fixed to be 20.

Table 5.1. Performance of the Upper Bounding Procedures on Small Instances

Instances	# of Binary Variables / Constraints	DAP		# of Binary / Continuous Variables / Constraints	ALA-CPX	
		CPU	UB		CPU	UB
chgn_4_8	2304/268	0,9	308,00	32/4/12	0,3	370,00
chgn_4_24	57600/2332	1172,5	1301,00	96/4/28	0,8	1627,26
chgn_5_20	42000/2025	7202,1	10276,00	100/5/25	2,3	12738,40
chgn_5_25	81250/3155	7204,8	14683,00	125/5/30	3,3	13570,83
chgn_5_30	139500/4535	7211,9	23769,00	150/5/35	3,2	17680,60
chgn_5_40	328000/8045	7238,2	30184,00	200/5/45	10,0	30855,40
chgn_5_50	637500/12555	7948,9	26829,00	250/5/55	3,7	25129,21
chgn_5_100a	5050000/50105	16644,8	41300,00	500/5/105	2,9	33357,60
chgn_5_100b	5050000/50105	15049,3	51197,00	500/5/105	6,8	36509,01
chgn_5_100c	5050000/50105	20186,3	53022,00	500/5/105	3,4	41149,67
chgn_5_100d	5050000/50105	15559,1	N/A	500/5/105	2,2	34677,63
chgn_5_100e	5050000/50105	20613,2	93653,00	500/5/105	1,2	37795,16
chgn_6_20	50400/2426	7202,9	14956,00	120/6/26	8,1	15160,69
chgn_6_25	97500/3781	7205,8	9647,00	150/6/31	2,5	12869,22
chgn_6_30	167400/5436	7214,6	N/A	180/6/36	7,8	16623,24
chgn_7_20	58800/2827	7203,1	N/A	140/7/27	2,7	9114,74
chgn_7_25	113750/4407	7216,4	26417,00	175/7/32	4,6	12015,09
chgn_7_30	195300/6337	7223,6	27256,00	210/7/37	22,1	18672,01
chgn_8_25	130000/5033	7209,9	N/A	200/8/33	7,9	14282,95
chgn_8_30	223200/7238	7226,5	N/A	240/8/38	43,4	16001,70
chgn_9_25	146250/5659	7210,8	N/A	225/9/34	4,8	12326,52
chgn_9_30	251100/8139	7233,4	N/A	270/9/39	51,7	15209,66
chgn_10_25	162500/6285	7227,2	N/A	250/10/35	7,0	10082,65
chgn_10_30	279000/9040	7276,0	N/A	300/10/40	151,8	15209,07
chgn_10_40	656000/16050	7797,6	N/A	400/10/50	120,2	15036,32
Average	**1162774/14643,9**	**8819,2**	**28319,87**	**14643,9/252,5/48,5**	**19,0**	**18722,59**

Table 5.2. Performance of the Upper Bounding Procedures on Small Instances (continued)

Instances	ALA – VLSN				ALA – TS			
	R-VLSN		RBM-VLSN		ALA-TS-wFC		ALA-TSwoFC	
	CPU	UB	CPU	UB	CPU	UB	CPU	UB
chgn_4_8	0,41	310,00	3,56	308	1,10	310,00	0,08	310,00
chgn_4_24	1,26	1315,29	4,33	1301	0,49	1519,53	0,50	1449,64
chgn_5_20	3,14	9748,97	5,36	9209	98,30	9748,97	1,17	11251,30
chgn_5_25	1,70	11095,37	6,11	9814	29,59	11208,09	0,51	13481,18
chgn_5_30	3,41	15885,64	6,02	15204	33,61	17047,73	0,66	21936,96
chgn_5_40	2,37	23449,20	9,13	22060	25,20	25106,00	0,98	30977,85
chgn_5_50	7,81	20904,44	10,74	19678	4,45	22164,13	1,56	27558,89
chgn_5_100a	7,14	32109,45	28,16	29889	10,07	38738,45	8,95	36910,20
chgn_5_100b	10,40	31859,11	29,15	30239	20,03	37297,84	13,56	36639,58
chgn_5_100c	9,35	38659,01	27,81	35376	17,12	42352,63	8,08	45857,72
chgn_5_100d	15,00	37637,78	26,15	33767	16,19	41889,98	9,41	44388,53
chgn_5_100e	13,69	39759,75	28,27	37087	22,11	43300,90	9,00	45702,60
chgn_6_20	1,19	10717,01	7,16	10420	250,27	10385,93	5,21	12480,01
chgn_6_25	1,25	9132,02	6,12	7895	47,17	9361,01	0,62	14426,94
chgn_6_30	1,77	15502,34	6,44	14574	69,40	16545,72	1,11	22469,06
chgn_7_20	5,79	8726,38	6,95	8187	438,90	8726,38	2,18	9141,02
chgn_7_25	1,75	10027,00	6,43	9822	141,52	10069,08	1,23	14046,06
chgn_7_30	2,65	14784,96	7,8	14528	110,03	15892,08	1,95	19685,14
chgn_8_25	4,12	9272,31	11,21	9048	515,66	9455,46	4,84	14042,21
chgn_8_30	2,18	14354,66	9,28	13378	234,73	16891,04	3,17	21157,13
chgn_9_25	1,15	8916,00	10,65	9521	355,92	9196,02	10,28	13613,02
chgn_9_30	6,36	12811,97	12,32	11973	446,75	14616,22	2,61	19845,67
chgn_10_25	5,77	7033,02	15,04	6213	321,52	8736,01	19,05	9944,96
chgn_10_30	7,68	14620,06	10,77	12675	403,54	15583,21	15,85	17698,10
chgn_10_40	4,39	15980,30	13,36	14245	239,11	17633,35	6,26	23868,13
Average	**4,87**	**16584,48**	**12,33**	**15456,44**	**154,11**	**18151,03**	**5,15**	**21155,28**

Table 5.3. Performance of the Upper Bounding Procedures on Large Instances

Instances	ALA – VLSN				ALA – TS			
	R-VLSN		RBM-VLSN		ALA-TSwFC		ALA-TSwoFC	
	CPU	UB	CPU	UB	CPU	UB	CPU	UB
chgn_10_100	26,1	45046,93	45,8	40691,06	108,1	52597,61	8,7	64559,21
chgn_20_50	63,7	19970,06	203,7	15865,03	522,8	20166,04	14,5	22376,15
chgn_20_250	140,5	68670,43	158,6	68048,86	403,3	97657,17	201,4	114118,71
chgn_25_250a	121,6	72083,02	254,0	69637,64	339,6	9485,46	159,2	103842,13
chgn_25_250b	96,5	81006,58	329,5	79616,47	494,6	131657,34	70,5	149052,31
chgn_25_250c	165,8	73937,26	214,6	70665,13	386,3	98711,34	151,1	104877,89
chgn_25_250d	184,2	63204,11	226,8	62705,45	830,0	86841,62	87,8	109839,10
chgn_25_250e	168,7	81029,13	238,7	79277,41	615,3	101650,11	78,4	115153,77
chgn_30_100	18,4	31855,92	43,9	26440,64	767,7	48515,61	30,0	54017,28
chgn_50_250	221,2	48766,91	288,3	48217,84	583,7	61229,54	182,7	101735,64
Average	**120,7**	**58557,04**	**200,4**	**56116,55**	**505,1**	**70851,18**	**98,4**	**93957,22**

Table 5.4. The Average Computational Results on Small Instances

Algorithm	Version	CPU	UB	%
DAP	**DAP**	8819,19	28319,87	83,22%
ALA-CPX	**ALA-CPX**	19,04	18722,59	21,13%
ALA-VLSN-R	**R-VLSN**	**4,87**	16584,48	7,30%
ALA-VLSN-RBM	**RBM-VLSN**	12,33	**15456,44**	0,00%
ALA-TS-wFC	**TS with FC**	154,11	18151,03	17,43%
ALA-TS-woFC	**TS without FC**	5,15	21155,28	36,87%

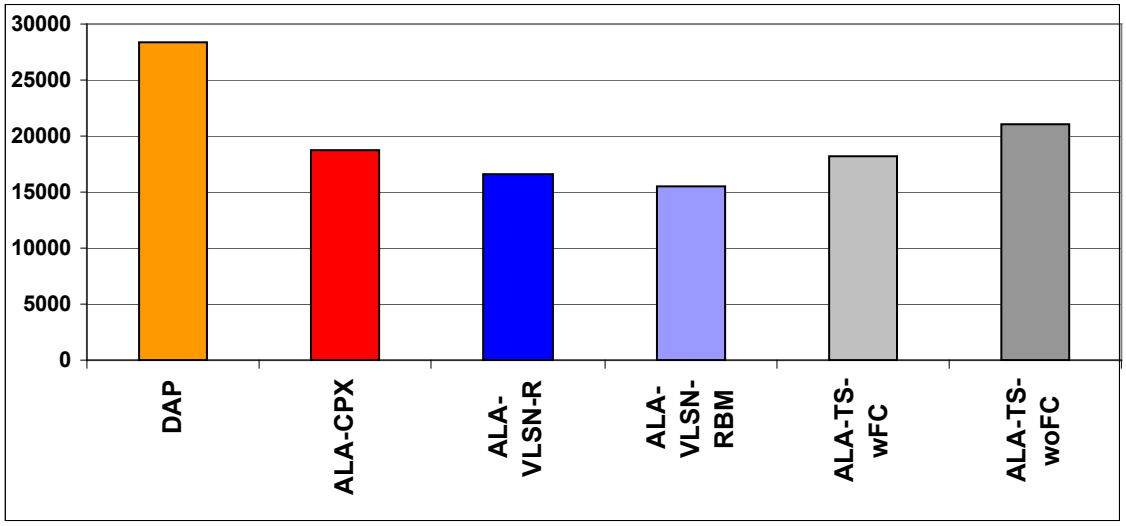

Figure 5.1. The Average Upper Bound Values Obtained with the Proposed Algorithms on Small Instances

Table 5.5. Performance of the TS Algorithm with the Increasing CPU Time

Instances	ALA-TSwoFC									
	(200 / 50)		(500 / 125)		(1000 / 250)		(2000 / 500)		(3000 / 750)	
	CPU	UB	CPU	UB	CPU	UB	CPU	UB	CPU	UB
chgn_4_8	0,1	310,00	1,71	310,00	3,40	310,00	6,81	310,00	10,14	310,00
chgn_4_24	0,5	1372,49	11,44	1375,08	21,61	1340,01	43,70	1340,01	65,98	1373,81
chgn_5_20	1,3	10808,20	11,99	9652,68	22,76	10040,82	60,37	9748,97	65,81	10029,81
chgn_5_25	0,9	14736,28	15,12	12712,71	35,89	11873,56	58,68	13273,13	88,73	10268,11
chgn_5_30	1,1	17620,56	22,67	17658,97	41,66	17556,72	84,82	17777,16	139,84	17030,96
chgn_5_40	2,24	29718,34	35,29	29110,06	72,75	27580,44	153,62	26146,13	204,67	25360,02
chgn_5_50	2,48	24840,54	56,08	24129,87	106,68	23607,79	207,00	23422,90	308,22	23954,94
chgn_5_100a	10,62	37272,11	229,87	35377,34	383,48	35663,95	761,28	36532,71	1129,54	34984,08
chgn_5_100b	14,11	36822,71	205,51	36306,83	388,71	35915,19	900,97	35822,90	1242,79	35519,08
chgn_5_100c	13,26	43626,64	191,52	43024,63	379,21	41073,22	884,37	40181,15	1143,60	40140,14
chgn_5_100d	12,50	41811,05	218,86	40594,64	404,46	40847,17	762,54	41119,98	1202,11	39837,70
chgn_5_100e	11,30	44861,44	204,68	43322,71	383,37	43011,80	829,24	39453,00	1781,88	42041,04
chgn_6_20	2,58	12040,61	24,30	10909,89	39,76	11412,88	80,01	11022,30	121,21	10909,89
chgn_6_25	0,96	10412,09	17,31	10636,62	34,52	10722,77	67,76	10543,14	121,17	10225,49
chgn_6_30	1,28	18874,44	24,42	18393,54	49,38	16907,55	99,10	19059,88	151,74	16486,97
chgn_7_20	5,42	9092,39	21,76	8228,39	57,26	8228,39	94,64	8228,39	136,20	8228,39
chgn_7_25	1,68	12936,04	25,48	11367,05	45,46	11169,72	94,90	10696,75	128,45	11036,07
chgn_7_30	1,58	19189,15	29,19	19051,36	60,81	18080,10	120,86	16865,90	171,47	18149,92
chgn_8_25	6,61	10973,78	42,76	11064,16	66,75	10683,02	140,25	11121,24	227,05	10151,76
chgn_8_30	3,05	18516,62	34,64	15873,86	74,48	17041,54	153,85	15688,08	220,91	14110,06
chgn_9_25	9,08	11868,03	49,02	12354,01	87,30	9806,02	184,56	10099,84	302,39	9667,02
chgn_9_30	5,40	17139,74	45,64	17630,86	83,59	17220,75	174,18	16065,77	264,11	14763,64
chgn_10_25	15,22	7660,02	69,33	7768,67	135,37	7288,41	281,62	6838,02	438,36	7441,02
chgn_10_30	7,82	17309,44	59,29	17702,05	115,46	16169,57	226,22	15812,28	411,21	15547,07
chgn_10_40	6,24	24765,59	84,18	21121,52	139,98	19722,07	285,69	20180,24	422,01	19188,80
Average	**5,50**	**19783,13**	**69,28**	**19027,10**	**129,36**	**18530,94**	**270,28**	**18294,00**	**419,98**	**17870,23**

6. CONCLUSION

In this work, we have addressed the rectilinear distance SSCMWP which aims to find the optimal location of a number of facilities that will serve a set of customers with known locations in the continuous plane under the single source assumption in order to minimize the total transportation cost which is proportional to the customer demands and distances between facilities and customers. We assume that the distances between facilities and customers are rectilinear. To the best of our knowledge, we have encountered with only two studies on the SSCMWP in the literature. These two studies are focused on the Euclidean distance SSCMWP. We have proposed four upper bounding approaches for the SSCMWP. These are DAP heuristic, ALA-CPX heuristic, ALA-VLSN heuristic and ALA-TS heuristic. We have devised two versions of the ALA-VLSN algorithm. They are which consider random ejection rule (ALA-VLSN-R) and the one which works with the RBM rule (ALA-VLSN-RBM).

Two versions of the ALA-TS heuristic are developed. These are ALA-TS heuristic with feasibility chasing phase (ALA-TS-FC) and ALA-TS heuristic without feasibility chasing phase (ALA-TSwoFC). We have also proposed the DAP heuristic which requires excessive CPU time. DAP heuristic is the least efficient method for obtaining upper bounds on the SSCMWP. According to our experiments, we have observed that both ALA-VLSN and ALA-TS heuristics yield outstanding performance.

The computational results indicate that both R-VLSN and RBM-VLSN heuristics are accurate and efficient for the rectilinear distance SSCMWP and competitive with other well-known algorithms. According to our computational experiments, comparing all methods, ALA-VLSN heuristic yields significantly better results in reasonable CPU times. Thus, considering the trade-off between accuracy and efficiency we can also

recommend ALA-VLSN heuristic as a reasonable solution approach. In average VLSN heuristics clearly outperform TS approach.

Comparing four heuristics, ALA-CPX heuristic is the worst one. Even if the computational time of the ALA-CPX heuristic is acceptable, its accuracy is not good. The accuracy of ALA-CPX in the final solution depends on its initial solution where the facility locations are determined randomly in the initialization step of the heuristic.

As a further research avenue, we believe that combining VLSN search strategy with other search schemes such as TS or SA has yield more efficient and accurate upper bounding heuristics for the rectilinear SSCMWP. Moreover, many other metaheuristics such as GA and Greedy Randomized Adaptive Search Procedure (GRASP) can also be implemented.

REFERENCES

[1] Weber, A., "Theory of the Location of Industries", Chicago, III., *The University of Chicago Presss*, (1909).

[2] Cooper, L., "The Transportation-Location Problem", *Operations Research,* 20 (1), 94–108, (1972).

[3] Sherali, H.D., Nordai, F.L., "NP-Hard, Capacitated, Balanced p-Median Problems on a Chain Graph with a Continuum of Link Demands", *Mathematics of Operations Research*, 13, 32–49, (1988).

[4] Foulds, L.R., Wilson, J.M., "A Variation of the Generalized Assignment Problem Arising in the New Zealand Dairy Industry", *Annals of Operations Research*, 69, 105–114, (1997).

[5] Akyüz, M.H., Öncan, T., Altınel, İ.K., "Efficient Approximate Solution Methods for the Multi-Commodity Capacitated Multi-Facility Weber Problem", *Computers and Operations Research* , 39 (2), 225–237, (2012).

[6] Akyüz, M.H., Öncan, T., Altınel, İ.K., "The Multi-Commodity Capacitated Multi-Facility Weber Problem: Heuristics and Confidence Intervals", *IIE Transactions*, 42(12), 825–841, (2010).

[7] Akyüz, M.H., Öncan, T., Altınel, İ.K., "Efficient Lower and Upper Bounds for the Multi-Commodity Capacitated Multi-Facility Weber Problem with Rectilinear Distances", In: Voss, S. , Pahl, J. , Schwarse, S., editors. *Logistik Management Systeme Methoden Integration Berlin, Heidelberg*: Springer-Verlag, 229–245, (2009).

[8] Akyüz, M.H., Öncan, T., Altınel, İ.K., "The Multi-Commodity Capacitated Multi-Facility Weber Problem: Heuristics and Confidence Intervals", In: Ao, SI., Castillo, O., Douglas, C., Dagan Feng, D., Lee, J-A., editors. *Proceedings of the International Multi Conference of Engineers and Computer Scientists*, IMECS, vol. II, 2042– 2047,(2009).

[9] Rosing, K.E., "An Optimal Method for Solving the (Generalized) Multi-Weber Problem", *European Journal of Operational Research*, 58 (3), 414–426, (1992).

[10] Krau, S., "Extensions du Problème de Weber", *PhD Thesis*, Ecole Polytechnique de Montréal, (1997).

[11] Cooper, L., "Heuristic Methods for Location-Allocation Problems", *Siam Review*, 6 (1), 37–53, (1964).

[12] Hansen, P., Mladenović, N., Taillard, E., "Heuristic Solution of the Multisource Weber Problem as a *p*-Median Problem", *Operations Research Letters*, 22 (2-3), 55–62, (1998).

[13] Brimberg, J., Hansen, P., Mladenović, N., Taillard, E.D., "Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem", *Operations Research*, 48 (3), 444–460, (2000).

[14] Gamal, M.D.H., Salhi, S., "A Cellular Type Heuristic for the Multisource Weber Problem", *Computers and Operations Research*, 30 (11), 1609–1624, (2003).

[15] Salhi, S., Gamal, M.D.H., "A GA-Based Heuristic for the Multi-Weber Problem", *Annals of Operations Research*, 123, 203–222, (2003).

[16] Taillard, E.D., "Heuristic Methods for Large Centroid Clustering Problems", *Journal of Heuristics*, 9 (1), 51–73 , (2003).

[17] Brandeau, M.L., Chiu, S.S., "Sequential Location and Allocation: Worst Case Performance and Statistical Estimation", *Location Science*, 1 (4), 289–298, (1993).

[18] Aras, N., Özkısacık, K.C., Altınel, İ.K., "Solving the Uncapacitated Multi-Facility Weber Problem by Vector Quantization and Self-Organizing Maps", *Journal of the Operational Research Society*, 57, 82–93, (2006).

[19] Liu, C.M., Kao, R.L., Wang, A.H., "Solving Location-Allocation Problems with Rectilinear Distances by Simulated Annealing", *Journal of the Operational Research Society*, 45, 1304–1315, (1994).

[20] Lozano, S., Guerrero, F., Onieva L., Larraneta, J., "Kohonen Maps for Solving a Class of Location-Allocation Problems", *European Journal of Operational Research*, 108, 106–117, (1998).

[21] Hsieh, K.H., Tien, F.C., "Self-Organizing Feature Maps for Solving Location-Allocation Problems with Rectilinear Distances", *Computers and Operations Research*, 31, 1017–1031, (2004).

[22] Sherali, H.D., Ramachandran, S., Kim, S., "A Localization and Reformulation Discrete Programming Approach for the Rectilinear Distance Location-Allocation Problem", *Discrete Applied Mathematics*, 49 (1-3), 357–278, (1994).

[23] Al-Loughani, L., "Algorithmic Approaches for Solving the Euclidean Distance Location-Allocation Problems", *PhD Dissertation*, Blacksburg, Virginia: Industrial and System Engineering, Virginia Polytechnic Institute and State University, (1997).

[24] Sherali, H.D., Al-Loughani, I., Subramanian, S., "Global Optimization Procedures for the Capacitated Euclidean and l_p Distance Multifacility Location-Allocation Problem", *Operations Research*, 50, 433–448, (2002).

[25] Sherali, H.D., Shetty, C.M., "The Rectilinear Distance Location-Allocation Problem", *Journal of Regional Science*, 16, 309–315, (1975).

[26] Sherali, H.D., Tunçbilek, C.H., "A Squared-Euclidean Distance Location-Allocation Problem", *Naval Research Logistics*, 39, 447–469, (1992).

[27] Aras, N., Altınel, İ.K., Orbay, M., "New Heuristic Methods for the Capacitated Multi-facility Weber Problem", *Naval Research Logistics*, 54, 21–32, (2007).

[28] Aras, N., Orbay, M., Altınel, İ.K., "Efficient Heuristics for the Rectilinear Distance Capacitated Multi-Facility Weber Problem", *Journal of the Operational Research Society*, 59, 64–79, (2008).

[29] Aras, N., Yumuşak, S., Altınel, İ.K., "Solving the Capacitated Multi-Facility Weber Problem by Simulated Annealing Threshold Accepting and Genetic Algorithms", In: Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W.J., Hartl, R.F., Reimann, M., editors. *Metaheuristics: progress in complex systems optimization, Berlin*: Springer, 91–112, (2007).

[30] Zainuddin, Z.M., Salhi, S., "A Perturbation-Based Heuristic for the Multisource Weber Problem", *European Journal of Operational Research*, 179, 1194–1207, (2007).

[31] Luis, M., Said, S., Nagy, G., "A Guided Reactive GRASP for the Capacitated Multi-Source Weber Problem", *Computers and Operations Research*, 38 (7), 1014–1024, (2011).

[32] Luis, M., Salhi, S., Nagy, G., "Region-rejection Based Heuristics for the Capacitated Multi-source Weber Problem", *Computers and Operations Research*, 36, 2007–2015, (2009).

[33] Gong, D., Gen, M., Yamazaki, G., Xu, W., "Hybrid Evolutionary Method for Capacitated Location-Allocation Problem", *Computers and Industrial Engineering*, 33 (3-4), 577–580, (1997).

[34] Manzour-al-Ajdad, S.M.H., Torabi, S.A., Eshghi, K. "Single-Source Capacitated Multi-Facility Weber Problem - An Iterative Two Phase Heuristic Algorithm", *Computers and Operations Research*, 39, 1465-1476, (2012).

[35] Rönnqvist, M., Tragantalerngsak, S., Holt, J., "A Repeated Matching Heuristic for the Single-Source Capacitated Facility Location Problem", *European Journal of Operational Research*, 116, 51–68, (1999).

[36] Pirkul, H., Jayaraman, V., "A Multi-commodity, Multi-Plant, Capacitated Facility Location Problem formulation and Efficient Heuristic Solution", *Computers and Operations Research*, 25, 869–878, (1998).

[37] Darby-Dowman, K., Lewis, H.S., "Lagrangian Relaxation and the Single Source Capacitated Facility Location Problem", *Journal of the Operational Research Society,* 39, 1035–1040, (1988).

[38] Cortinhal, M.J., Captivo, M.E., "Upper and Lower Bounds for the Single Source Capacitated Location Problem", *European Journal of Operational Research,* 151 (2), 333–351, (2003).

[39] Tragantalerngsak, S., Holt, J., Rönnqvist, M., "An Exact Method for the Two-Echelon, Single Source, Capacitated Facility Location Problem", *European Journal of Operational Research*, 123 (3), 473–489 (2000).

[40] Tragantalerngsak, S., Holt, J., Rönnqvist, M., "Lagrangian Heuristics for the Two-Echelon, Single-Source, Capacitated Facility Location Problem", *European Journal of Operational Research,* 102, 611–625 (1997).

[41] Ghiani, G., Guerriero, F., Musmanno, R., "The Capacitated Plant Location Problem with Multiple Facilities in the Same Time", *Computers and Operations Research*, 50, 268–274, (1999).

[42] Chen, C.H., Ting, C.J., "Combining Lagrangian Heuristic and Ant Colony System to Solve the Single Source Capacitated Facility Location Problem", *Transportation Research Part E*, 44(1), 1099–1122, (2008).

[43] Chen, C.H., Ting, C.J., "Applying Multiple Ant Colony System to Solve the Single Source Capacitated Facility Location Problem", *Lecture Notes in Computer Science*, 4150, 508–509, (2006).

[44] Bornstein, C.T., Azlan, H.B., "The Use of Reduction Tests and Simulated Annealing for the Capacitated Location Problem", *Location Science*, 6, 67–81, (1998).

[45] Lin, C.K.Y., "Stochastic Single-Source Capacitated Facility Location Model with Service Level Requirements", *International Journal of Production Economics*, 117, 439–451, (2009).

[46] Singhtaun, C., Charnsethikul, P., "Efficient Heuristics for Single-Source Capacitated Multi-Facility Weber Problems", *Applied Operations Research*, Proceedings of the 38th International Conference on Computers and Industrial Engineering, 35–39, (2008).

[47] Brimberg, J., Hansen, P., Mladenović, N., Salhi, S., "A Survey of Solution Methods for the Continuous Location-Allocation Problem", *International Journal of Operations Research*, 5 (1), 1–12, (2008).

[48] Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F., "A Very Large-Scale Neighbourhood Search Algorithm for the Multi-Resource Generalized Assignment Problem", *Discrete Optimization*, 1, 87–98, (2004).

[49] Minic, S.M., Punnen, A.P., “Local Search Intensified: Very Large-Scale Variable Neighborhood Search for the Multi-Resource Generalized Assignment Problem”, *Discrete Optimization*, 6, 370–377, (2009).

[50] Ahuja, R.K., Orlin, J.B., Sharma, D., “A Composite Very Large-Scale Neighbourhood Structure for the Capacitated Minimum Spanning Tree Problem”, *Operations Research Letters*, 31, 185–194, (2003).

[51] Ergun, Ö., Orlin, J.B., “A Dynamic Programming Methodology In Very Large Scale Neighborhood Search Applied to the Traveling Salesman Problem”, *Discrete Optimization*, 3, 78–85, (2006).

[52] Ahuja, R.K., Jha, K.C., Orlin, J.B., Sharma, D., “Very Large-Scale Neighbourhood Search for the Quadratic Assignment Problem”, *INFORMS Journal on Computing*, 19 (4), 646–657, (2007).

[53] Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P., “A Survey of Very Large-Scale Neighbourhood Search Techniques”, *Discrete Applied Mathematics*, 123, 75–102, (2002).

[54] Öncan, T., Kabadi, S.N., Nair, K.P.K., Punnen, A.P., “VLSN Search Algorithms for Partitioning Problems Using Matching Neighbourhoods”, *Journal of the Operational Research Society*, 59, 388–398, (2008).

[55] Ahuja, R.K., Orlin, J.B., Pallottino, S., Scaparra, M.P., Scutellá, M.G., “A Multi-Exchange Heuristic for the Single-Source Capacitated Facility Location Problem”, *Management Science*, 50 (6), 749–760, (2004).

[56] Holberg, K., Ronnqvist, M., Yuan, D., “An Exact Algorithm for the Capacitated Facility Location Problems with Single Sourcing”, *European Journal of Operational Research*, 113, 544–559 (1999).

[57] Hindi, H., Pienkosz, K., “Efficient Solution of Large Scale, Single-Source, Capacitated Plant Location Problem”, *Journal of the Operational Research Society*, 50, 268–274, (1999).

[58] Martello, S., Toth, P., “Knapsack Problems: Algorithms and Computer Implementations”, *Wiley Series in Discrete Mathematics and Optimization*, New York, (1990).

[59] Thisse, J.F., Ward, J.E., Wendell, R.E., “Some Properties of Location Problems with Block and Round Norms”, *Operations Research*, 32 (6), 1309–1327, (1984).

[60] Wendell, R.E., Hurter, A.P., “Location Theory, Dominance and Convexity”, *Operations Research*, 21, 314–320, (1973).

[61] Hansen, P., Perreur, J., Thisse, F., “Location Theory, Dominance and Convexity: Some Further Results”, *Operations Research*, 28(5), 1241–1250, (1980).

[62] Weiszfeld, E., “Sur le Point Lequel la Somme des Distances de n Points Donnés est Minimum”, *Tôhoku Mathematical Journal*, 43, 355–386, (1937).

[63] Brimberg, J., Love, R.F., “Global Convergence of a Generalized Iterative Procedure for the Minimum Location Problem with l_p Distances”, *Operations Research*, 41, 1153–1163, (1993).

[64] Taillard, E.D., “Parallel Iterative Search Methods for Vehicle Routing Problems”, *Networks*, 23, 661–673, (1993).

APPENDIX : C++ Codes

// THESE CODES BELONG TO ONLY ONE VERSION OF AN ALGORITM: RBM-VLSN

```
#include "resource.h" #include <iostream> #include <stdlib.h> #include
<cstdio> #include <stdio.h> #include <math.h> #include <fstream>
#include <time.h> #include <string> #include <ostream> #include
<Windows.h>
#define INF 100000000
#define ITNUMBERTWOPHASE 100
#define MAKSZAMAN 10000
#define WEISITNUMBER 5000
#define MAKSDENEME 10000
#define FEASIBLEDENEME 100
#define DENEMEBIPARTITEALTKISIM 50
#define MODSAYISI 3
#define IYISONUCCIKMADI  20
using namespace std;
//MAIN
int main (int argc, char **argv)
{
      clock_t start, finish;
      start = clock();
      int i,j,m,facility,boyut ;
      char *dosyaismi=new char[128];
         if(argc>0)
               dosyaismi=argv[1];
      double kobay;
        ifstream girdi1;
        girdi1.open(dosyaismi);
        if(!girdi1)
            exit(1);
          girdi1>>kobay;
            facility=(int)kobay;
```

```
            girdi1>>kobay;
            boyut=(int)kobay;
            girdi1.close();
      double cputime, oandakizaman=0;
      double *x=new double[boyut];
      double *y=new double[boyut];
      double *w=new double[boyut];
      double *wyeni=new double[boyut] ;
      double **uzaklik=new double*[facility];
               for(int i=0;i<facility;i++)
                        uzaklik[i]=new double[boyut];
      double *capacite=new double[facility];
      double **atama=new double *[facility];
                  for(i=0;i<facility;i++)
                        atama[i]=new double[boyut];
      double *xson=new double[facility];
      double *yson=new double[facility];
      double iterasyonmaliyet=100000000, minimummaliyet=100000000,
ortalama;
      int itmaliyetsayaci=0,sayactoplami=0,sayachesap,
sayac3,altsayac;;
      double **atamatut=new double *[facility];
                  for(i=0;i<facility;i++)
                        atamatut[i]=new double[boyut];
      double **atamaeniyi=new double *[facility];
                  for(i=0;i<facility;i++)
                        atamaeniyi[i]=new double[boyut];
   double
bipartite2maliyet,altbipartite2maliyet,altkobaymaliyet,kobaymaliyet;
        ifstream girdi2;
        girdi2.open(dosyaismi);
        if(!girdi2)
            exit(1);
          girdi2>>kobay; girdi2>>kobay; girdi2>>kobay;
            for(i=0;i<boyut;i++)
            {
              girdi2>>kobay; girdi2>>kobay;
              x[i]=kobay;
              girdi2>>kobay;
```

```
            y[i]=kobay;
          }
          girdi2>>kobay;
                for(i=0;i<facility;i++)
                      for(j=0;j<boyut;j++)
                            girdi2>>kobay;
          for(i=0;i<facility;i++)
          {
           girdi2>>kobay;
           capacite[i]=kobay;
          }
          for(j=0;j<boyut;j++)
          {
           girdi2>>kobay;
           w[j]=kobay;
          }
          girdi2.close();
        srand(time (NULL));
for(m=0;m<ITNUMBERTWOPHASE && oandakizaman<MAKSZAMAN;m++)
 {
          ilkatama(facility,boyut,atama);
          if((boyut/facility)>=11)
     bipartitematchingustsinirli(facility,boyut,atama,w,capacite,saya
chesap,MAKSDENEME,FEASIBLEDENEME,0.3);
          else if((boyut/facility)>=7)
     bipartitematchingustsinirli(facility,boyut,atama,w,capacite,saya
chesap,MAKSDENEME,FEASIBLEDENEME,0.4);
          }
          else
     bipartitematchingustsinirli(facility,boyut,atama,w,capacite,saya
chesap,MAKSDENEME,FEASIBLEDENEME,1);
          sayactoplami=sayactoplami+sayachesap;
          ortalama=(double)(sayactoplami/(m+1));
          for(i=0;i<facility;i++)
          {
                for(j=0;j<boyut;j++)
                {
                      if(atama[i][j]==0)
                            wyeni[j]=0;
```

```
                if(atama[i][j]==1)
                        wyeni[j]=w[j];
            }
weisfeld(x,y,wyeni,boyut,WEISITNUMBER,xson[i],yson[i]);
      }
      for(i=0;i<facility;i++)
            for(j=0;j<boyut;j++)
      uzaklik[i][j]=rectlindistance(x[j],y[j],xson[i],yson[i]);
      sayac3=0;
      kobaymaliyet=100000000;//yüz milyon
      while(sayac3<facility*boyut)
      {
            altsayac=0;
            altkobaymaliyet=100000000;//yüz milyon
            matristutma(facility,boyut,atamatut,atama);
            while(altsayac<DENEMEBIPARTITEALTKISIM)
            {
            bipartitematchingallocationimpr(facility,boyut,atamat
      ut,w,capacite,uzaklik,MODSAYISI);
            altbipartite2maliyet=amacfonkdegeri(atamatut,uzaklik,
      w,facility,boyut);
               if(altbipartite2maliyet<altkobaymaliyet)
               {
               matristutma(facility,boyut,atamaeniyi,atamatut);
                     altkobaymaliyet=altbipartite2maliyet;
                     matristutma(facility,boyut,atamatut,atama);
               }
                     altsayac=altsayac+1;
            }
            bipartite2maliyet=altkobaymaliyet;
            matristutma(facility,boyut,atama,atamaeniyi);
            if(bipartite2maliyet<kobaymaliyet)
            {
                  kobaymaliyet=bipartite2maliyet;
                        for(i=0;i<facility;i++)
                        {
                              for(j=0;j<boyut;j++)
                              {
                                    if(atama[i][j]==0)
```

```
                                        wyeni[j]=0;
                                    if(atama[i][j]==1)
                                        wyeni[j]=w[j];
                                }
weisfeld(x,y,wyeni,boyut,WEISITNUMBER,xson[i],yson[i]);
                            }
                                for(i=0;i<facility;i++)
                                    for(j=0;j<boyut;j++)
        uzaklik[i][j]=rectlindistance(x[j],y[j],xson[i],yson[i]);
                                sayac3=0;
                    }
                    sayac3=sayac3+1;
            }
                            for(i=0;i<facility;i++)
                            {
                                for(j=0;j<boyut;j++)
                                {
                                    if(atama[i][j]==0)
                                        wyeni[j]=0;
                                    if(atama[i][j]==1)
                                        wyeni[j]=w[j];
                                }
                        wendellhurter(x,y,wyeni,boyut,xson[i],yson[i]);
                                }
                                for(i=0;i<facility;i++)
                                    for(j=0;j<boyut;j++)
        uzaklik[i][j]=rectlindistance(x[j],y[j],xson[i],yson[i]);
iterasyonmaliyet=amacfonkdegeri(atama,uzaklik,w,facility,boyut);
            if(iterasyonmaliyet<minimummaliyet && iterasyonmaliyet>0)
            {
                    itmaliyetsayaci=0;
                    minimummaliyet=iterasyonmaliyet;
            }
            else
                    itmaliyetsayaci=itmaliyetsayaci+1;
            if(itmaliyetsayaci>IYISONUCCIKMADI)
                    goto atla;
            finish = clock();
            oandakizaman = (double)(finish-start)/CLOCKS_PER_SEC;
```

```
 }
      atla:
        finish = clock();
        cputime = (double)(finish-start)/CLOCKS_PER_SEC;
        ofstream cikti;
        cikti.open("sonuclarRBMVLSN.txt",ios::app);
        cikti.precision(10);
        if(!cikti)
            exit(1);
        cikti<<dosyaismi<<"Cpu Suresi:"<<cputime<<"
Maliyet:"<<minimummaliyet<<"Ort. Matching:"<<ortalama<<"Iterasyon
Sayisi:"<<m<<endl;
        cikti.close();
      delete [] x;
      delete [] y;
      delete [] w;
      delete [] wyeni;
      delete [] capacite;
      delete [] xson;
      delete [] yson;
      for (i=0; i<facility; i++){ delete uzaklik[i];}delete []
uzaklik;
      for (i=0; i<facility; i++){ delete atama[i];}delete [] atama;
      return 0;
}
//FONCTIONS
int indexyeni(int I,int i, int j)
{ return (I*i)+j; }
void matristutma(int facility,int boyut,double **matris1,double
**matris2)
{
      int i,j;
      for(i=0;i<facility;i++)
            for(j=0;j<boyut;j++)
                  matris1[i][j]=matris2[i][j];
}
void hungarian(int size, double **Cost_Matrix,double &cost, int
*&Solution)
{
```

http://ranger.uta.edu/~weems/NOTES5311/hungarian.c

```
}
int feasiblekontrol(double **atama, double *w, double *capacite, int
facility, int boyut)
{
  int i,j,sayac=0;
  double kisit;
    for(i=0;i<facility;i++)
    {
      kisit=0;
           for(j=0;j<boyut;j++)
           kisit=kisit+(atama[i][j]*w[j]);
             if(kisit<=capacite[i])
             sayac=sayac+1;
    }
      if(sayac==facility)
            return 1;
      else
            return 0;
}
double amacfonkdegeri(double **atama, double **distance, double *w,int
facility,int boyut)
{
  double sonuc=0,aradeger;
  int i,j;
  for(i=0;i<facility;i++)
  {
      aradeger=0;
      for(j=0;j<boyut;j++)
            aradeger=aradeger+(atama[i][j]*distance[i][j]*w[j]);
      sonuc=sonuc+aradeger;
  }
  return sonuc;
}
void weisfeld(double *xkoor, double *ykoor, double *weight,int
boyut,int itnumber,double &sonnoktax,double &sonnoktay)
{
int i, j;
  double toplamx=0,toplamy=0,toplamw=0;
```

```
  double toplamx2=0,toplamy2=0,toplamw2=0;
  double xilk, yilk, xiki, yiki;
  for(i=0;i<boyut;i++)
  toplamw=toplamw+weight[i];
  for(i=0;i<boyut;i++)
  {
  toplamx=toplamx+(weight[i]*xkoor[i]);
  toplamy=toplamy+(weight[i]*ykoor[i]);
  }
  xilk=(double)toplamx/toplamw;
  yilk=(double)toplamy/toplamw;
  for(j=0;j<itnumber;j++)
  {
     double kokici,*g=new double[boyut];

        for(i=0;i<boyut;i++)
        {
            if(xilk==xkoor[i]  && yilk==ykoor[i])
              g[i]=weight[i];
            else{
                kokici=(double)(xilk-xkoor[i])*(xilk-xkoor[i])+(yilk-
ykoor[i])*(yilk-ykoor[i]);
                g[i]=(double)weight[i]/(sqrt(kokici));
                }
        }
       toplamx2=0,toplamy2=0,toplamw2=0;
       for(i=0;i<boyut;i++)
       toplamw2=toplamw2+g[i];
       for(i=0;i<boyut;i++)
       {
       toplamx2=toplamx2+(xkoor[i]*g[i]);
       toplamy2=toplamy2+(ykoor[i]*g[i]);
       }
    if(toplamw2==0)
      {
          xiki=xilk ;
          yiki=yilk;
      }
    else
```

```
      {
          xiki=toplamx2/toplamw2;
          yiki=toplamy2/toplamw2;
      }
      double dongukosulu1,dongukosulu2;
      dongukosulu1=fabs(xiki-xilk);
      dongukosulu2=fabs(yiki-yilk);
     if(xiki==xilk && yiki==yilk)
       {
             sonnoktax=xiki;
             sonnoktay=yiki;
             break;
       }
     if(dongukosulu1<(0.0001) && dongukosulu2<(0.0001))
       {
             sonnoktax=xiki;
             sonnoktay=yiki;
             break;
       }
            if(j==itnumber-1)
              {
             sonnoktax=xiki;
             sonnoktay=yiki;
              }
            delete [] g;
  }
}
void wendellhurter(double *xkoor, double *ykoor, double *weight,int
boyut,double &sonnoktax,double &sonnoktay)
{
   int i,m,k, j,farklixsayisi,farkliysayisi,aynicikti,sayi;
   double totalcost,kobaymaliyet=10000000;//on milyon
   double *xsakla=new double[boyut], *ysakla=new double[boyut];
   for(i=0;i<boyut;i++)
         xsakla[i]=xkoor[i];
   farklixsayisi=1;
   for(i=0;i<boyut;i++)
   {
        aynicikti=1;
```

```
      if(weight[i]!=0)
      {
          for(j=0;j<i;j++)
          {
                if(xkoor[i]==xsakla[j])
                {
                      aynicikti=0;
                      xsakla[i]=-1;
                }
          }
      }
      else
            xsakla[i]=-1;
      farklixsayisi=farklixsayisi+aynicikti;
}
for(i=0;i<boyut;i++)
       ysakla[i]=ykoor[i];
farkliysayisi=1;
for(i=0;i<boyut;i++)
{
      aynicikti=1;
      if(weight[i]!=0)
      {
          for(j=0;j<i;j++)
          {
                if(ykoor[i]==ysakla[j])
                {
                      aynicikti=0;
                      ysakla[i]=-1;
                }
          }
      }
      else
            ysakla[i]=-1;
      farkliysayisi=farkliysayisi+aynicikti;
}
   double *adayyerx=new double[(farklixsayisi*farkliysayisi)];
   double *adayyery=new double[(farklixsayisi*farkliysayisi)];
   sayi=0;
```

```
      for(k=0;k<boyut;k++)
      {
            for(i=0;i<farkliysayisi;i++)
            {
                  if(xsakla[k]!=-1)
                  {
                      adayyerx[sayi]=xsakla[k];
                        sayi=sayi+1;
                  }
            }
      }
      sayi=0;
      for(k=0;k<farklixsayisi;k++)
      {
            for(m=0;m<boyut;m++)
            {
                  if(ysakla[m]!=-1)
                  {
                        adayyery[sayi]=ysakla[m];
                        sayi=sayi+1;
                  }
            }
      }
      for(i=0;i<farklixsayisi*farkliysayisi;i++)
      {
            totalcost=0;
            for(j=0;j<boyut;j++)
            {
            totalcost=totalcost+weight[j]*(fabs(xkoor[j]-
adayyerx[i])+fabs(ykoor[j]-adayyery[i]));
            }
            if(totalcost<kobaymaliyet)
            {
                  sonnoktax=adayyerx[i];
                  sonnoktay=adayyery[i];
                  kobaymaliyet=totalcost;
            }
      }
   delete [] xsakla;
```

```
   delete [] ysakla;
   delete [] adayyerx;
   delete [] adayyery;
}
double rectlindistance(double xcord,double ycord,double xnoktasi,
double ynoktasi)
{
   double distance;
   distance=fabs(xcord-xnoktasi) + fabs(ycord-ynoktasi);
   return distance;
}
void ilkatama(int facility,int boyut,double **atama)
{
    int i,j,k,t,tut=0;
    double toplam,toplam2,maks=0;
    double *satirtoplamlari=new double[facility];
    double butuntoplam=0;
        for(i=0;i<facility;i++)
        {
              for(j=0;j<boyut;j++)
              {
                    toplam=0;
                    toplam2=0;
                    atama[i][j]=rand()%2;
                    for(k=i-1;k>=0;k--)
                              toplam=toplam+atama[k][j];
                        if(toplam>0)
                              atama[i][j]=0;
                        if(i==facility-1 && toplam==0)
                              atama[i][j]=1;
                  if(j==boyut-1)
                  {
                    for(k=0;k<boyut;k++)
                    {
                    toplam2=toplam2+atama[i][k];
                    satirtoplamlari[i]=toplam2;
                      if(toplam2>maks)
                        {
                        maks=toplam2;
```

```
                    tut=i;
                    }
                }
            }
            if(satirtoplamlari[i]==0)
            {
                for(k=0;k<boyut;k++)
                {
                  if(atama[tut][k]==1)
                    {
                    atama[tut][k]=0;
                    atama[i][k]=1;
                    maks=maks-1;
                    satirtoplamlari[i]=1;
                    satirtoplamlari[tut]=satirtoplamlari[tut]-1;
                    break;
                    }
                }
            }
            for(t=0;t<i;t++)
            {
              if(satirtoplamlari[t]>maks)
              {
                maks=satirtoplamlari[t];
                tut=t;
              }
            }
          }
        }
        delete [] satirtoplamlari;
}
void bipartitematchingustsinirli(int facility,int boyut,double
**atama, double *w,double *capacite,int &sayachesap,int maksdeneme,int
feasibledeneme,double ustsinir)
{
            int i,j, sayi,rassal, a,b,umut,sayac,genelsayac ;
            int *atamasayisisol=new int[facility];
            int *atamasayisisag=new int[facility ;
            double **maliyet=new double *[facility];
```

```
            for(i=0;i<facility;i++)
                  maliyet[i]=new double[facility];
double **atamatut=new double *[facility];
            for(i=0;i<facility;i++)
                  atamatut[i]=new double[boyut];
double toplam1,toplam2,totalcost;
int *sonuc=new int[facility];
matristutma(facility,boyut,atamatut,atama);
sayac=0;
genelsayac=0;
birdaha:
      for(i=0;i<facility;i++)
            atamasayisisol[i]=0;
      for(i=0;i<facility;i++)
         for(j=0;j<boyut;j++)
           if(atama[i][j]==1)
                  atamasayisisol[i]=atamasayisisol[i]+1;
      for(i=0;i<facility;i++)
      {
            gel:
            atamasayisisag[i]=rand()%(atamasayisisol[i]+1);
if(atamasayisisag[i]>(ustsinir*atamasayisisol[i]))
                  goto gel;
      }
int **soltaraf=new int *[facility];
            for(i=0;i<facility;i++)
                  soltaraf[i]=new int[atamasayisisol[i]];
int **sagtaraf=new int *[facility];
            for(i=0;i<facility;i++)
                  sagtaraf[i]=new int[atamasayisisag[i]];
      for(i=0;i<facility;i++)
      {
                  sayi=0;
                  for(j=0;j<boyut;j++)
                  {
                        if(atama[i][j]==1)
                        {
                              soltaraf[i][sayi]=j;
                              sayi=sayi+1;
```

```
                    }
                }
}
for(i=0;i<facility;i++)
{
  for(j=0;j<atamasayisisag[i];j++)
  {
            yeniden:
    rassal=rand()%atamasayisisol[i];
            if(soltaraf[i][rassal]==-1)
                  goto yeniden;
      sagtaraf[i][j]=soltaraf[i][rassal];
      soltaraf[i][rassal]=-1;
  }
}
for(i=0;i<facility;i++)
{
      toplam1=0;
      for(a=0;a<atamasayisisol[i];a++)
            if(soltaraf[i][a]!=-1)
                  toplam1=toplam1+w[soltaraf[i][a]];
            for(j=0;j<facility;j++)
            {
                  toplam2=0;
                  for(b=0;b<atamasayisisag[j];b++)
      toplam2=toplam2+w[sagtaraf[j][b]];
      maliyet[i][j]=capacite[i]-(toplam1+toplam2);
                  if(maliyet[i][j]<0)
                        maliyet[i][j]=10000;
            }
}
      hungarian(facility,maliyet,totalcost,sonuc);
for(i=0;i<facility;i++)
      for(j=0;j<boyut;j++)
            atama[i][j]=0;
for(i=0;i<facility;i++)
{
  for(j=0;j<atamasayisisol[i];j++)
        if(soltaraf[i][j]!=-1)
```

```
                              atama[i][soltaraf[i][j]]=1;
                        for(umut=0;umut<atamasayisisag[sonuc[i]];umut++)
                              if(sagtaraf[sonuc[i]][umut]!=-1)
                                    atama[i][sagtaraf[sonuc[i]][umut]]=1;
                  }
                  for (i=0; i<facility; i++)
{ delete soltaraf[i];}delete [] soltaraf;
                  for (i=0; i<facility; i++)
{ delete sagtaraf[i];}delete [] sagtaraf;
                  sayac=sayac+1;

      if(feasiblekontrol(atama,w,capacite,facility,boyut)!=1 &&
sayac<maksdeneme)
                        goto birdaha;
                  sayachesap=sayac;
                  if(sayac>=maksdeneme)
                  {
                        for(i=0;i<facility;i++)
                              for(j=0;j<boyut;j++)
                                    atama[i][j]=atamatut[i][j];
                        genelsayac=genelsayac+1;
                        if(genelsayac>feasibledeneme)
                        {
                              exit(1);
                        }
                        sayac=0;
                        goto birdaha;
                  }
      delete [] atamasayisisol;
      delete [] atamasayisisag;
      delete [] sonuc;
      for (i=0; i<facility; i++){ delete atamatut[i];}delete []
atamatut;
      for (i=0; i<facility; i++){ delete maliyet[i];}delete []
maliyet;
}
void bipartitematchingallocationimpr(int facility,int boyut,double
**atama, double *w,double *capacite,double **distance,int modsayi)
{
```

```
int i,j,sayi,rassal,a,b,umut;
int *atamasayisisol=new int[facility];
int *atamasayisisag=new int[facility];
double **maliyet=new double *[facility];
            for(i=0;i<facility;i++)
                  maliyet[i]=new double[facility];
double toplam1,toplam2,toplam3,toplam4,totalcost;
int *sonuc=new int[facility];
for(i=0;i<facility;i++)
      atamasayisisol[i]=0;
for(i=0;i<facility;i++)
   for(j=0;j<boyut;j++)
     if(atama[i][j]==1)
              atamasayisisol[i]=atamasayisisol[i]+1;
for(i=0;i<facility;i++)
{
      if(atamasayisisol[i]>0)
      {
            tekrarla:
            atamasayisisag[i]=rand()%modsayi + 1;
            if(atamasayisisag[i]>atamasayisisol[i])
                  goto tekrarla;
      }
      else
            atamasayisisag[i]=0;
}
int **soltaraf=new int *[facility];
      for(i=0;i<facility;i++)
            soltaraf[i]=new int[atamasayisisol[i]];
int **sagtaraf=new int *[facility];
      for(i=0;i<facility;i++)
            sagtaraf[i]=new int[atamasayisisag[i]];
for(i=0;i<facility;i++)
{
            sayi=0;
            for(j=0;j<boyut;j++)
            {
                  if(atama[i][j]==1)
                  {
```

```
                                soltaraf[i][sayi]=j;
                                sayi=sayi+1;
                        }
                }
        }
        for(i=0;i<facility;i++)
        {
          for(j=0;j<atamasayisisag[i];j++)
          {
                    yeniden:
            rassal=rand()%atamasayisisol[i];
                    if(soltaraf[i][rassal]==-1)
                           goto yeniden;
              sagtaraf[i][j]=soltaraf[i][rassal];
              soltaraf[i][rassal]=-1;
          }
        }
        for(i=0;i<facility;i++)
        {
              toplam1=0;
              for(a=0;a<atamasayisisol[i];a++)
                    if(soltaraf[i][a]!=-1)
                          toplam1=toplam1+w[soltaraf[i][a]];
                    for(j=0;j<facility;j++)
                    {
                          toplam2=0;
                          for(b=0;b<atamasayisisag[j];b++)
                                toplam2=toplam2+w[sagtaraf[j][b]];
                    maliyet[i][j]=capacite[i]-(toplam1+toplam2);
                          if(maliyet[i][j]!=0)
                                maliyet[i][j]=1000000;
                          else
                          {
                                toplam3=0;
                                toplam4=0;
                                for(a=0;a<atamasayisisag[i];a++)
toplam3=toplam3+(distance[i][sagtaraf[i][a]]*w[sagtaraf[i][a]]);
                                for(b=0;b<atamasayisisag[j];b++)
toplam4=toplam4+(distance[i][sagtaraf[j][b]]*w[sagtaraf[j][b]]);
```

```
                                    maliyet[i][j]=toplam4-toplam3;
                              }
                        }
            }
            for(i=0;i<facility;i++)
                  for(j=0;j<boyut;j++)
                        atama[i][j]=0;
            for(i=0;i<facility;i++)
            {
              for(j=0;j<atamasayisisol[i];j++)
                    if(soltaraf[i][j]!=-1)
                        atama[i][soltaraf[i][j]]=1;
              for(umut=0;umut<atamasayisisag[sonuc[i]];umut++)
                    if(sagtaraf[sonuc[i]][umut]!=-1)
                        atama[i][sagtaraf[sonuc[i]][umut]]=1;
            }
      for (i=0; i<facility; i++){ delete soltaraf[i];}delete []
soltaraf;
      for (i=0; i<facility; i++){ delete sagtaraf[i];}delete []
sagtaraf;
      delete [] atamasayisisol;
      delete [] atamasayisisag;
      delete [] sonuc;
      for (i=0; i<facility; i++){ delete maliyet[i];}delete []
maliyet;
}
```

BIOGRAPHICAL SKETCH

M. Emre DEMİRCİOĞLU was born in İstanbul on December 25, 1984. He has studied at Büyükşehir Hüseyin Yıldız Anatolian High School where he graduated in 2002. He started his undergraduate studies at Industrial Engineering Department of Galatasaray University in 2003.

During his education at Galatasaray University, with his two schoolmates, he has participated in the logistics case competition organized by LODER (Logistics Association) and they have obtained the first ranking. In 2009, he obtained the B.S. degree in Industrial Engineering as a sixth ranking graduate and he enrolled in Graduate School at University Galatasaray.

Since November 2009, he has been working as a research assistant at Industrial Engineering Department of Galatasaray University.