ABSTRACT

BUCCI, MICHAEL JAMES. Solution Procedures for Logistics Network Models with Economies of Scale. (Under the direction of Michael G. Kay and Donald P. Warsing.)

As supply chains have become more dynamic the difference in the time horizon for strategic decisions has diminished, resulting in supply chains that are more flexible with no/low fixed facility costs. This trend requires the development of solution approaches that can combine the traditionally separate strategic, tactical, and operational decisions in an integrative manner, incorporating a range of decision variables and cost considerations while producing good, and possibly near-optimal, solutions in reasonable time.

This research begins to address these issues through the development of heuristic approaches to solve large-scale facility location problems that reflect economies of scale in the per-unit costs of processing goods and/or holding safety stock of those goods to protect against uncertainty in demand. Such non-linear economies of scale are well known in practice, but are often excluded or overly simplified in location models due to their non-linear nature. Combining and extending existing heuristic approaches, we develop and analyze several meta-heuristics to solve a location problem with a non-linear, concave cost function, which is used as a surrogate for more computationally complex cost functions. The resulting solution methods offer near-optimal solutions with relatively modest computational effort. These meta-heuristics are then applied to a focused study on the use of approximations to represent safety stock inventory costs in location models. This research evaluates the commonly used "Square Root Law" and a more general concave cost function against the explicit safety stock inventory calculation in models with and without inter-customer demand correlation. The results highlight the conditions for which these functions accurately approximate actual inventory costs and/or when they generate location solutions that are close to those generated by the explicit computation of inventory levels. The meta-heuristics are then applied to a reverse logistics location problem for the carpet

industry. This application requires us to recommend locations for processing used carpet in a setting where the recycling facilities to be located exhibit economies of scale in processing. We use our modeling approach to analyze an existing, smaller-scale used carpet collection network and also evaluate a larger hypothetical national collection network, providing insight into the number of recycling facilities that should be located and their respective size. We compare the results of formulating and solving models with and without economies of scale, highlighting the value of their inclusion on the results.

Solution Procedures for Logistic Network Design Models with Economies of Scale

by
Michael James Bucci

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Industrial Engineering

Raleigh, North Carolina

2009

APPROVED BY:

_____	_____
Michael G. Kay Committee Co-Chair	Donald P. Warsing Committee Co-Chair
_____	_____
Jeffrey A. Joines	Reha Uzsoy

BIOGRAPHY

Michael J. Bucci was born in Endicott, NY. He graduated from Union-Endicott High School in 1986, and then obtained an undergraduate degree in Ceramic Engineering at Alfred University in 1990. Mike then worked for one year before returning to study for his master's degree in Industrial Engineering at Binghamton University (SUNY) under the supervision of Dr. R.E. Emerson. From here, Mike worked in industry for nine years in a variety of engineering and management positions. He then returned to academia to pursue a degree in Industrial and Systems Engineering at North Carolina State University under the supervision of Dr. Michael Kay and Dr. Donald Warsing. Mike and his wife are the proud parents of three wonderful children.

ACKNOWLEDGEMENTS

I would like to thank the following people for their generous and continued support for this work:

- Dr. Michael Kay and Dr. Donald Warsing served as my advisors and co-chairs. Both provided invaluable support and guidance to me throughout this journey.
- Dr. Reha Uzsoy and Dr. Jeffrey Joines served as my committee members, and I thank them for all their support and guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Appendix A.1.2

CHAPTER 1: Introduction

Introduction

As supply chains have become more dynamic the difference in the time horizon for strategic decisions has diminished, resulting in more flexible supply chains with no/low fixed facility costs. This trend necessitates the development of solution approaches that can evaluate the traditionally separate strategic, tactical, and operational decisions in an integrative manner, incorporating a range of decision variables and cost considerations while producing good, and possibly near-optimal, solutions in reasonable time. This research studies these types of facility location–allocation problems by extending traditional facility location models and their associated solution techniques. The solution methods envisioned for these types of models combine multiple techniques such as metaheuristics, simulation, and optimization methods into a flexible framework to solve problems of realistic practical size.

Dissertation Outline

Chapters 2, 3, and 4 are structured as self-standing journal submissions. Each of these chapters is formatted to the requirements of the journal for which they were submitted or are intended to be submitted. Chapter 5 provides conclusions and directions for future work for the entire dissertation. The Appendices include supplemental information for chapters two, three, and four that were not suitable to include in the journal publications. This includes original data, computer programs, and additional observations and findings. Additionally, two short conference paper submissions are given in Appendix A. These papers were a precursor to the more detailed journal paper displayed in Chapter 2.

Overview of Chapters

Chapter 2 investigates facility location models that incorporate economies of scale in unit costs of production and/or inventory holding. These models have known customer locations with deterministic demands, and the objective is to locate an unknown number of

distribution centers to minimize transportation costs and facility costs, the latter being composed of fixed costs and variable costs that are non-linear in the number of units processed. To solve these models several metaheuristic solution approaches were developed by combining and extending existing techniques. These solution approaches were compared in an extensive empirical study with the results, highlighting the combinations of techniques that can find near-optimal solutions with moderate computational effort. Since our objective is to assess the ability to solve problems with objective functions that are more complex to calculate, new measures of computational effort are developed.

Chapter 3 investigates methods for modeling risk-pooling effects associated with centralizing safety stocks in large-scale facility location models. These models have known customer locations with stochastic demands, and the objective is to locate an unknown number of distribution centers to minimize the transportation and safety inventory costs. The "Square Root Law" and a more general concave cost function, which accounts for facility size more explicitly, are compared with the explicit calculation of safety stock inventory on the basis of total solution cost, solution time, and solution structure. Models with no correlation in demand across customer locations are studied along with models that include inter-customer demand correlation. The latter effort also resulted in a new method for generating large scale matrices for inter-customer correlation in demand. The results from models with no inter-customer correlation of demand show that the "Square Root Law" is fairly inaccurate at estimating inventory costs due to the assumptions underlying this rule, while the more general concave cost function always outperforms the "Square Root Law" with minimal additional computational effort. In models with low and moderate inter-customer correlation of demand, both approximation techniques poorly estimate the inventory required, suggesting the need to use the explicit calculation or the development of new approximation methods.

Chapter 4 investigates an industry application: a reverse logistics network for recycling used carpet. The model attempts to minimize the total cost to locate an unknown number of carpet recycling facilities that process used carpet collected from an established network of collection points. The model includes transportation cost from the collection facility to the recycling facility, fixed facility costs at the recycling facility, and non-linear processing costs at the recycling facility, the latter exhibiting economies of scale as the facility size increases. The model evaluates a known carpet collection network and a hypothetical collection network that assumes a significant increase in collection locations and collection rates to meet carpet industry recycling targets. We show that economies of scale have a significant impact on the solution structure, and also demonstrate the impact that collection volumes have on the number of facilities in the network.

CHAPTER 2: Metaheuristics for Facility Location Problems with Economies of Scale

Michael J. Bucci[*], Michael G. Kay[*], Donald P. Warsing[†], Jeffrey A. Joines[**]
[*]**Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA**
[†]**Department of Business Management, North Carolina State University, Raleigh, NC 27695, USA**
[**]**Department of Textile Engineering/Chemistry/Science, North Carolina State University, Raleigh, NC 27695, USA**

Reprint of paper submitted to the European Journal of Operational Research (EJOR)

Metaheuristics for Facility Location Problems with Economies of Scale

Michael J. Bucci[*], Michael G. Kay[*], Donald P. Warsing[†], Jeffrey A. Joines[**]
[*]Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA
[†]Department of Business Management, North Carolina State University, Raleigh, NC 27695, USA
[**]Department of Textile Engineering/Chemistry/Science, North Carolina State University, Raleigh, NC 27695, USA

Abstract

We develop solution methods for solving large-scale facility location problems that include non-linear costs, reflecting economies of scale in unit costs of production and/or safety stock inventory. Through an extensive empirical study we compare several metaheuristic solution methods that combine algorithmic construction, allocation, and location techniques, including alternate location-allocation (ALA), variable neighborhood search, and tabu search. We evaluate the solution approaches with respect to not only solution quality and time, but also with new measures of computational effort that offer insight on the applicability of the proposed methods to problems with more complex objective functions. While no one solution approach is dominant in terms of computational effort and solution quality, we show that selectively combining methods in a metaheuristic framework can provide near-best solutions with relatively low computational effort.

Keywords

Supply chain management, facility location, metaheuristics, logistics, economies of scale

1. Introduction

We develop metaheuristic solution methods for large scale facility location problems that include non-linear costs that reflect economies of scale in unit costs of production and/or inventory, the latter to due risk pooling. The model extends the well studied *P*-median and uncapacitated fixed charge facility location problem (UFLP) by including a simple nonlinear concave term in the objective function that represents economies of scale with respect to the size of each facility. This nonlinear function is used as a surrogate for more computationally complex functions as it allows the performance of a variety of different metaheuristics to be compared to optimal solutions. The solution methods combine and extend existing algorithmic construction, allocation, and location techniques, including alternate location-

allocation (ALA), variable neighborhood search, and tabu search. A large set of test instances is used in an empirical study to compare solution methods based on solution time, solution quality, and measures of computational effort. By measuring computational effort based on the number of computational evaluations required, we gain insight into the behavior of the heuristics that is not apparent from considering only solution time and objective function value. As computational evaluations are not typically measured within facility location models, we present a broad analysis of the different heuristics and their impact on these measurements, addressing the tradeoffs in solution quality (total cost) and computational effort.

The remainder of this paper is organized as follows. In Section 2 we review relevant literature while Section 3 describes the general model formulation. Our experimental design, solution approaches and computational results are explained in Section 4. Section 5 offers conclusions and directions for future research.

2. Literature Review

The multi-source Weber problem, [Brimberg et al. 2000], the *P*-median problem [Daskin 1995], and the uncapacitated fixed charge facility location problem (UFLP) [Mirchandani and Francis 1990] are three classic facility location models that have been widely studied [Daganzo 2005]. These models can be solved with exact methods, however, heuristics are used to solve most large instances as these problems have been shown to be NP hard [Hansen and Mladenovic 1997]. The alternate location-allocation (ALA) procedure proposed by Cooper [1963] is a widely cited iterative solution approach that many recent heuristics adapt to solve all three of these problems [Hansen and Mladenovic 1997, Bischoff and Dachert 2009]. A thorough survey and analysis of heuristics for the multi-source Weber problem has been conducted by Brimberg et al. [2000], who study the ALA procedure, tabu search (TS), variable neighborhood search (VNS), fixed neighborhood search (FNS), and *P*-median heuristic procedures. Mirchandani and Francis [1990] provide a review of discrete

location models and their use to represent continuous location problems. Daskin reviews well known add/drop/exchange heuristics for the *P*-Median and UFLP [Daskin 1995].

A variety of improvement techniques have been proposed for facility location-allocation solutions methods. Glover and Laguna [1997] review tabu search techniques and their applicability to metaheuristic solution methods and Gendreau and Potvin [2005] offer a more recent review of advances in tabu search. Hansen and Mladenovic [1997] describe several variants of Variable Neighborhood Search (VNS) and their use in solving large scale problems. Several other papers explore the application of VNS and Tabu Search to facility location problems [Resende and Werneck 2006, Melachovsky et al. 2005].

Several studies have extended facility location models to include nonlinear costs. Whitaker [1985] offers several flexible heuristics for an extension of the UFLP that includes nonlinear warehousing costs that are continuously concave over a range of possible warehouse sizes. These models differ from the *P*-median and UFLP in that a retailer may not be allocated to the closest facility, and there is interdependence between each customer allocation as the allocation impacts the facility costs [Daskin et al. 2002]. Bucci and Kay [2006] studied an extension of the multi-source Weber problem that included non-linear economies of scale. Their solution approach, an adaptation of the ALA procedure, was shown to behave similarly to the traditional ALA procedure for a range of production-transportation cost ratios and economies of scale values.

There have been several approaches to representing economies of scale in facility location models. Whitaker uses the following cost function for the i^{th} facility to represent the nonlinear warehousing costs in a problem with n customers

$$K_i \left(\sum_{j=1}^{n} D_j X_{ij} \right)^q \tag{2.1}$$

where K_i is a warehouse cost operator, D_j is the demand for customer j, X_{ij} is a binary variable set to 1 if warehouse i supplies node j, and q is a scale exponent, where q= –0.5 on the basis of inventory theory [Maister 1976]. A similar cost function is proposed by Rumelt [2001] for representing scale economies in manufacturing as a ratio, which takes the form

$$C_1 = \left\lfloor \; S_1 / S_2 \;^{b} \right\rfloor C_2 \tag{2.2}$$

where C_1 and C_2 are the unit production or processing costs in facilities of size S_1 and S_2, respectively, while b is the scale exponent (estimated to be –0.35 for manufacturing facilities). A piecewise linear approximation of the cost or combination of a fixed and linear cost term have been used to represent this non-linear behavior [Winston 2003, Perl and Daskin 1985]. Croxton and Zinn [2005] appraise the assumptions of the "Square Root Law" of inventory and its applicability to network design problems with aggregated product information. They propose using a discrete function to represent inventory costs as this method requires fewer constraints and variables relative to a piecewise linear approximation.

Solution time and closeness to optimal or best known solutions have been the primary means of comparing solution methods for facility location problems [Brimberg et al. 2000]. Although these two measures provide insight into the algorithms usefulness, they do not offer significant insight into the ability of the solution to solve models with increasingly complex cost functions. Hansen and Mladenovic [1997] used the number of iterations as a stopping condition for evaluating solution methods for the *P*-median problem. Bischoff and Dachert [2009] study a multi-connection location and allocation model and count the number of location-allocation "calls," using it as a stopping criteria and as a measure of the efficiency of different search methods. Simulation studies, which place a premium on the number of cycles required to reach a solution, frequently measure algorithm efficiency by measuring the number of computational cycles [Humphrey and Wilson 2000].

3 Model Formulation

The model formulation, an extension of the UFLP, is a slight variation of the model proposed by Whitaker [1985] and is defined as follows

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij} + \sum_{i=1}^{n} F_i Y_i + \sum_{i=1}^{n} C_i \;\; S_i \;\; S_i \tag{2.3}$$

subject to

$$\sum_{i=1}^{n} X_{ij} = 1 \text{ for } j = 1, 2, \ldots, n \tag{2.4}$$

$$S_i = \sum_{j=1}^{n} X_{ij} D_j \tag{2.5}$$

$$Y_i - X_{ij} \geq 0 \text{ for } i = 1, 2 \ldots, m \;\; j = 1, 2 \ldots, n \tag{2.6}$$

$$X_{ij} \in \; 0,1 \quad \text{for } i = 1, 2, \ldots, m \;\; j = 1, 2, \ldots, n \tag{2.7}$$

$$Y_i \in \; 0,1 \quad \text{for } i = 1, 2, \ldots, m \tag{2.8}$$

where

$X_{ij} =$ a binary variable set to 1 if warehouse i supplies node j

$Y_i =$ a binary variable set to 1 if warehouse i is open, and 0 otherwise

$C_{ij} =$ transportation cost from warehouse i to node j

$D_j =$ the demand at point (node) j ;

$F_i =$ the fixed operating cost of warehouse i;

$S_i =$ size of facility i

$C_i(S_i) =$ marginal unit cost for facility i as a function of its size, S_i

The first term in the objective function (2.3) computes the transportation cost from the facilities to the customers. The second term computes the fixed cost of locating facilities and the third term the marginal facility costs as a function of the facility size. Constraints (2.4) require that each retailer is served by only one warehouse while constraints (2.5) ensure that the facility size equals the customer demand allocated to the facility. Constraints

(2.6) state that a warehouse must be open ($Y_i = 1$) if it serves at least one retailer (X_{ij}=1 for some j=1,2,…,n). Constraints (2.7) and (2.8) are integer constraints.

The transportation cost is for one-way shipment and is calculated by multiplying a $/mile charge by the estimated road distance between the customer and facility locations. The fixed facility cost provides a proxy for the fixed construction/operating costs for the facility. The marginal facility costs represent nonlinear economies of scale for inventory or production as a function of facility size using (2.2). It is assumed there are no capacity constraints on facility size.

The number of facilities to locate is indeterminate at the outset and the marginal facility costs are approximated using a three-segment upper envelope piecewise linear approximation to allow for comparison to optimal solutions obtained using CPLEX 10.0.

4. Computational Results

This section will describe the experimental design, solution techniques, and computational results.

4.1 Experimental Design

The models represent a single-source single-tier retail distribution network with aggregated product information. The customer locations and the potential facility locations are the three-digit ZIP codes for the continental U.S. with non-zero population (877 in total). The geographical location of each three-digit ZIP code is taken to be the population centroid of its constituent five-digit ZIP codes. The customer weights are proportional to population densities. In generating the costs for the models we assume the supply chain mirrors existing retail networks in the U.S., which typically contain between five and thirty warehouses to serve most of the continental U.S. For example, Lowe's Companies [2006] and Walgreen Company [2006] respectively utilize 11 and 13 regional distribution centers to

serve the continental U.S. The great circle distance in statute miles, magnified by a circuity factor of 1.2, is used to determine the distance between facilities and customer locations [Ballou et al. 2002].

The following notation is used. Let $N = 1, 2, \ldots, 877$ be the (arbitrarily) ordered set of all customer locations (i.e., such that $|N| = 877$). For each problem instance that is modeled and solved, we define $N_n \subseteq N$ ($|N_n| = n$) to be the customer location set, with $n \in 60, 100, 400, 600, 877$ representing the instance sizes that are ultimately formulated and solved in this research. Further, let $M \subseteq N_n$ be the set of *m* candidate facility locations (i.e., $|M| = m$) for a particular problem. In all cases we use $M = N_n$ (such that $m = n$); however, in some subroutines we refer to instances with $M \subset N_n$ (such that $m < n$). Also, a range of scale exponents $b \in 0, -0.35, -0.5$ are tested for Equation (2.2).

We develop and assess solution methods through a series of empirical studies. The values of *b* that were tested (0, –0.35, and –0.5) were chosen, respectively, to represent no economies of scale, a typical manufacturing economy of scale, and the square root law for inventory. To evaluate a wide range of problems and parameters, smaller problems ranging from 60 – 600 customers were generated by sampling from the 877 customer problem set using a random weighted permutation with the customer demand normalized to maintain a total demand of 1.5 million units per year. The heuristic models were created and executed in MATLAB 7.2, utilizing Matlog as a building block for the models [Kay 2006]. CPLEX 10.0 was used to solve the integer and mixed integer formulations. All tests were run on a 3.2 GHz Intel Pentium 4 computer with 1527 MB of memory running Windows XP.

4.2 Metaheuristic Solution Approaches for Smaller Problems

To assess each heuristic we compare solution cost, solution time, and computational effort. We allow each heuristic to run to completion—until there is no improvement in total cost or until the program terminates due to the computer being out of memory (OoM). We developed three measures of computational effort. These measures count the number of computational evaluations needed to reach a solution. We consider these evaluation measurements to be more important measures of efficiency than solution time as they better predict the algorithm behavior as the complexity of the objective function increases. Each measure counts each evaluation of the nonlinear facility cost, computationally the most expensive cost to calculate in our problem formulation. As we discuss below, in an attempt to reduce the number of evaluations, a simplified cost function is utilized in some of the heuristic subroutines. These steps only include fixed facility costs and/or transportation costs.

We, characterize the evaluation measurements into three categories that count the number of objective function evaluations: construction evaluations, allocation evaluations, and location evaluations. An allocation evaluation is defined as each instance in which a customer allocation is attempted for each facility in the allocation subroutine. A construction evaluation is the same computationally as an allocation evaluation, except that it occurs in the ADD/construction subroutine. In some cases we report these two measures as a combined measure: however, we track them separately to gain a deeper understanding of the behavior of the heuristics. A location evaluation is each instance, within the location subroutine, in which the algorithm evaluates the re-location of a facility to another location.

We first test a set of heuristics on smaller sized instances with n=60 and 100. This set was reduced in the later empirical study to include those methods that offered the best combination of total cost and number of computational evaluations.. The heuristics tested

are summarized in Table 1. The first heuristic, AD-dALA, is an adaptation of the add/drop heuristics described by Daskin [1995] for the UFLP combined with a discrete ALA improvement procedure. In this approach the construction algorithm adds facilities considering only fixed facility costs and transportation costs to obtain a set of starting facility locations and set an upper bound on the number of facilities to locate. A greedy DROP procedure is then used to remove facilities from the solution until the total cost is no longer reduced.

Table 1: Metaheuristics Analyzed on Smaller Instances

Heuristic	Facility Add/ Construction	Customer Allocation	Facility Location
1. AD-dALA	Add with no scale costs, followed by Drop procedure with scale costs	All customers	Fixed Neighborhood Search (FNS) using Delaunay triangulation
2. MS -dALA	AD solution provides range in *p* values to test	Same as heuristic 1	Same as heuristic 1
3. W1	Greedy Add using all unused locations	Same as heuristic 1	FNS checking all locations in the subgraph
4. W2	Same as heuristic 3	Same as heuristic 1	Same as heuristic 1
5. W1-LAt	Same as heuristic 3	Only customers on convex hull of the current allocation	Same as heuristic 3, but only assess subgraphs with new facility allocations
6. W2- Lt	Same as heuristic 3	Same as heuristic 1	Same as heuristic 1, but only assess subgraphs with new facility allocations
7. W2-LAt	Same as heuristic 3	Same as heuristic 5	Same as heuristic 6
8. W1-S	Generate subset using simplified Whitaker ADD procedure from heuristic 3. Then use same procedure as heuristic 3	Same as heuristic 1	Same as heuristic 3
9. W2-S-LAt	Same as heuristic 8	Same as heuristic 5	Same as heuristic 6

Our DROP procedure is identical to that offered by Daskin, except in that our procedure incorporates the nonlinear facility costs into the DROP procedure by using the facility sizes

from the previous iteration? to determine the marginal facility unit cost. At the end of the DROP procedure, if one or more facilities have no retailers assigned to them, these facilities are removed to avoid a degenerate solution [Brimberg and Mladenovic 1999]. After the DROP procedure is complete, a discrete ALA improvement procedure is used. This procedure is an adaptation of the continuous location heuristic described by Bucci and Kay [2006]. Our discrete version differs from their heuristic by constraining the possible facility locations (m) to the set of customer locations, incorporating fixed facility location costs, and using a Delaunay triangulation (DT) discrete neighborhood relocation search in place of the continuous location search. Since the local neighborhood search may not yield the optimal facility locations for the allocation, the location procedure is re-run until there is no improvement in the solution.

Regarding the allocation procedure, the first allocation evaluation includes only transportation and fixed facility costs to allocate the customers to the facilities since we have not yet determined a facility size to calculate the marginal facility costs. Using this allocation, the marginal facility cost is then added to obtain a total cost for the current solution. In subsequent allocation cycles, the facility sizes from the previous allocation cycle are used to establish the marginal facility costs for the next allocation cycle with no updates to the facility costs within the allocation cycle. As in the location procedure, we re-run the allocation cycle until there is no further improvement in the solution. The location-allocation iterations continue until there is no improvement in total cost. To avoid a degenerative condition [Brimberg and Mladenovic 1999] we check for unused facilities after each allocation cycle and randomly relocate any unused facility prior to the next location cycle.

The second model, MS-dALA, is an adaptation of the d-eosALA procedure [Bucci et al. 2007] that uses the result of the ADD/DROP initialization procedure described above to determine a range for the number of facilities (p) to locate. Since this initialization may not provide the optimal p value we test the range $p - 2$ through $p + 1$ to improve the likelihood

we evaluate solutions with the optimal number of facilities. Using the results from Bucci and Kay [2006] and Houck et al. [1996], which provide guidance on the number of re-starts for random start ALA procedures, we perform forty random starts at each value of *p*. The discrete ALA procedure is identical to the procedure in the AD-dALA heuristic. At the end of the improvement procedure unused facilities are dropped from the solution. We use this MS-dALA procedure as a baseline for the other heuristics as it closely resembles the multi-start ALA procedure commonly used in facility location problems [Brimberg et al. 2000].

The third heuristic (W1) is a minor variant of the ADD algorithm described by Whitaker [1985]. Whitaker's heuristic is a modification of a greedy ADD procedure that includes an iterated location-allocation improvement procedure between each construction step. Adding a facility in each cycle is based first on the transportation and fixed cost; only afterwards are marginal facility costs added to establish a total cost for the solution. Like the d-eosALA heuristic, the W1 heuristic uses the facility sizes from the previous allocation cycle to determine the nonlinear facility costs for the current solution. The allocation procedure evaluates all retail customers in each cycle. The location procedure considers all customer locations in the subgraph as possible relocation sites for each facility. To reduce the possibility that the model will terminate prematurely, this process of ADD-allocate-locate continues until there are three solutions that are not better than the current best solution [Whitaker 1985]. Figure 1 depicts the general procedure of the W1 heuristic.

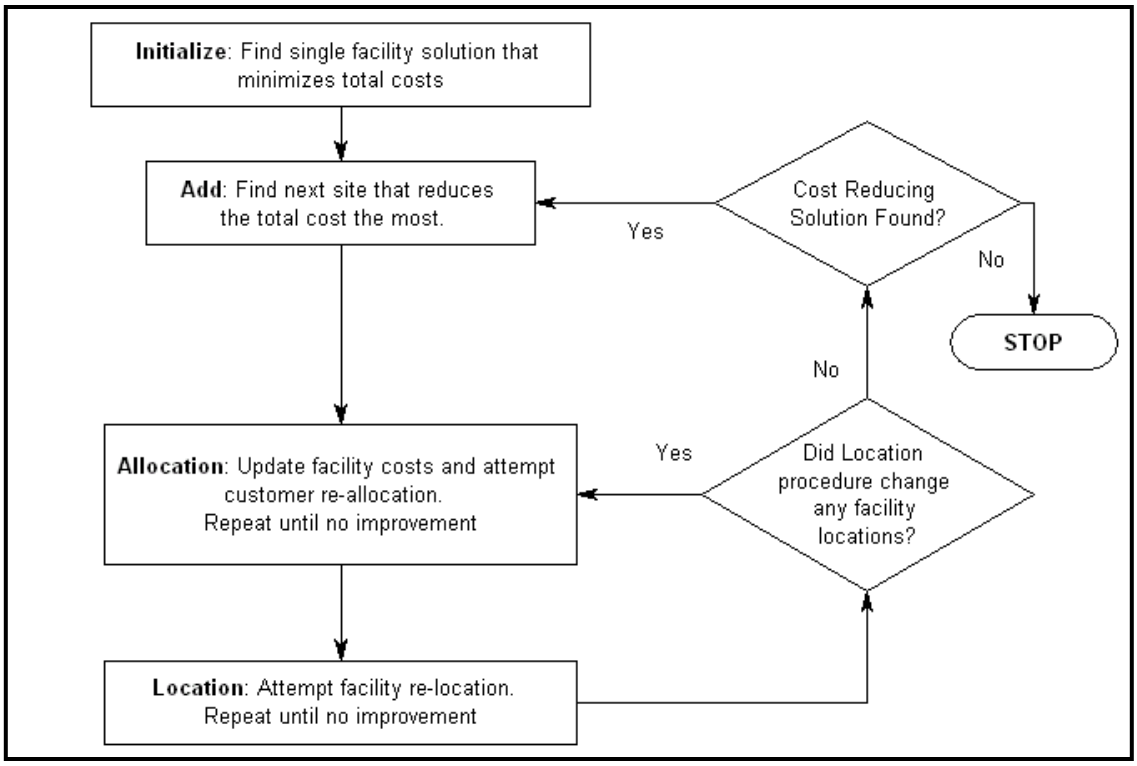

Figure 1: Solution Procedure for W1 Heuristic

Heuristics four to nine are modifications of the W1 procedure with changes in the ADD/construction, location, and/or allocation procedures in an attempt to reduce the computational evaluations without a significant degradation in the solution. The fourth heuristic (W2) differs from the W1 heuristic in utilizing the same Delaunay triangulation neighborhood search used in the AD-dALA heuristic. An example of the re-location set for one subgraph is shown in Figure 2. The locations labeled with either an asterisk or a dot (27 in total) are assessed in the W1 re-location search, while only the locations labeled with an asterisk (4 in total) are assessed in the W2 neighborhood re-location search.

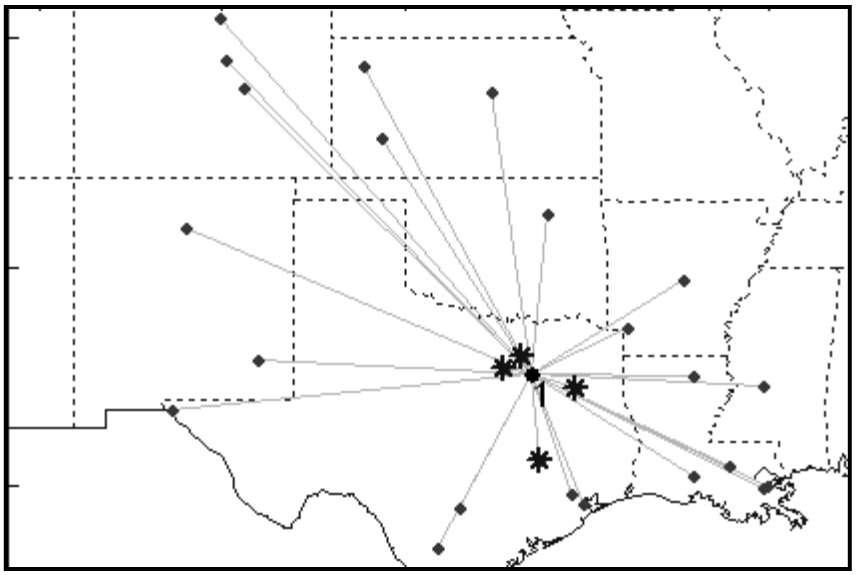

Figure 2: Example of a re-location set for the W1 and W2 Heuristics

The fifth heuristic (W1-LAt) differs from the W1 heuristic in both the location and allocation search. It was observed in early testing that the allocations for some facilities did not change in each location cycle, therefore, performing the re-location search for these facilities was unnecessary. To address this situation this heuristic incorporates a Tabu Search (TS) concept, short term memory, by storing the allocation from the previous cycle and only performs a re-location search in subgraphs in which the allocation has changed. In the allocation procedure only customers along the convex hull of the allocation are considered. An example of the re-allocation set for one subgraph is shown in Figure 3. The locations labeled with either a "x" or a dot (27 in total) are assessed in the W1 re-allocation search, while only the locations labeled with an asterisk (9 in total) are assessed in the W1-LAt convex hull re-allocation search. The W2-Lt heuristic adds the short term memory improvement to the W2 heuristic. The W2-LAt heuristic extends the W2-Lt heuristic by assessing customers only along the convex hull in the allocation procedure.

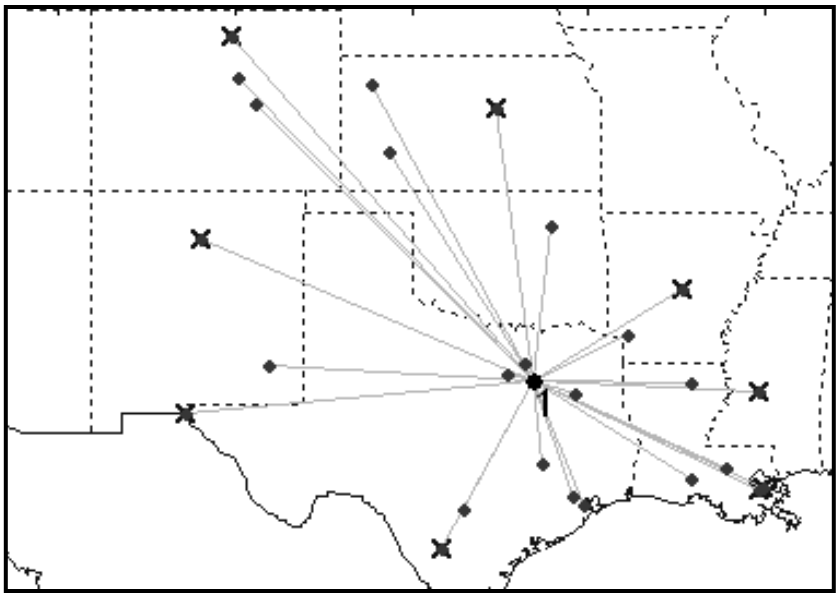

Figure 3: Example of a re-allocation set for the W1 and W1-LAt Heuristics

Heuristic 8 (W1-S) varies from the W1 heuristic only in the construction procedure. Rather than searching every unused facility location in the construction procedure, this heuristic only searches a subset of the unused facility locations, $M \subseteq N_n$ (such that $m \le n,\ m \le 100$). We use the W1 heuristic with a simplified objective function containing only transportation and fixed facility costs in an attempt to intelligently create the subset locations. The subset size m_s is defined by the following. The simplified W1 heuristic runs until there is no improvement in total cost and $m_s \ge 0.25n$, however, the size of m_s is capped at $m_s \le 100$. We set $m_s \le 100$ based on the clustering analysis research from Ballou [1994] who studied a similar customer location dataset. We set $m \ge 0.25n$ based on limited results showing this percentage did not exhibit a noticeable increase (0.001%) in the objective function. Figure 4 provides an example of the construction subset for a problem with n = 200 and m_s = 50. The locations with either a "x" or a dot is the complete possible facility set, while the locations with a "x" are in the subset. The final heuristic, W2-S-LAt, applies the subset in the construction procedure to the W2-LAt heuristic.

To improve the evaluation of the heuristics CPLEX 10.0 was used where possible to find optimal solutions. Early tests showed CPLEX could not find an optimal solution for problems with $n \geq 100$ using a three segment piecewise approximation; however, with b = 0.0 and the problem formulated as an integer program (IP) CPLEX quickly finds an optimal solution. Therefore, for problems with $n \geq 100$ CPLEX was used as a comparison for the heuristics for all problems with b = 0.0.

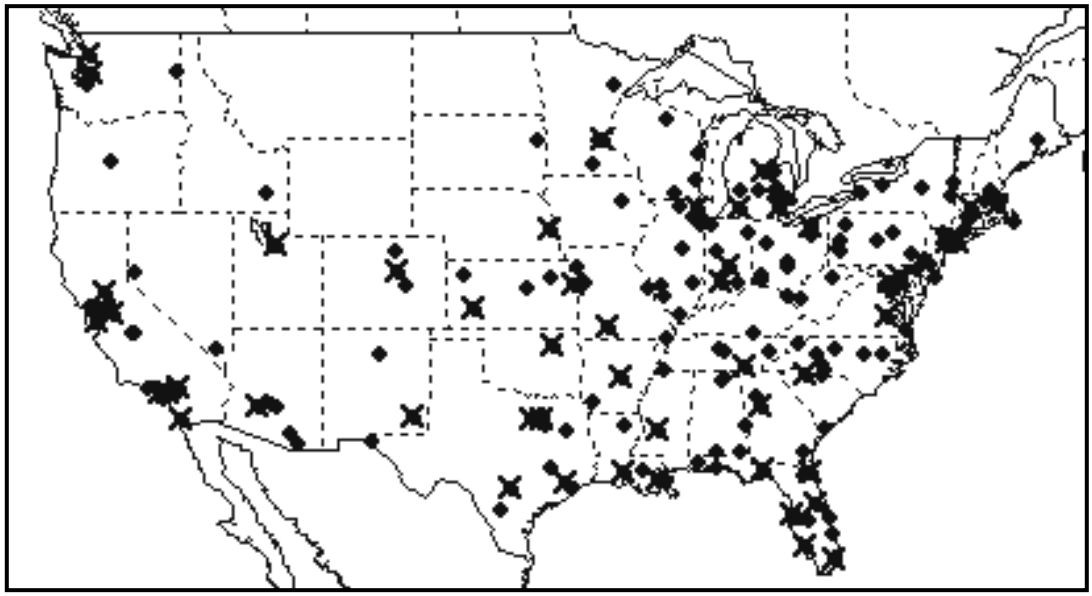

Figure 4: Example of construction subset for n = 200 and m_s = 50

4.3. Performance of Metaheuristics on Smaller Problems

Given that the dataset is new, we use the MS-dALA procedure as an indicator of the difficulty of the problem as multi-start procedures are a common baseline comparison for facility location problems [Houck et al. 1996, Brimberg et al. 2000]. We define a "good" solution to be less than 1% from the best solution found over all algorithms used. For n=100, b= –0.35 the MS-dALA procedure did not find any solutions within 2% of the best solution and on average was 11% from the best solution, thus allowing us to conclude the problems are significantly challenging to solve within 1% of the best known value.

For each test instance, we compute total cost, solution time, and the three measures of computational effort. The total cost was analyzed using a "percent above best" (%AB) measurement that has been used in facility location research [Brimberg et al. 2000]. Solution time and several evaluation measurements were analyzed using a "multiple from best" (xFB) measurement which is calculated by taking the heuristic value and dividing it by the best value found among all of the heuristics. This offers an intuitive relative comparison of solution time and number of evaluations required for the algorithms. A value of 1.0 means the heuristic achieved the best value.

Initial tests were conducted with $n \in \{60, 100\}$ using a random weighted permutation of the 877 customer dataset where weights were proportional to population densities. Ten problems were generated at each value of n and with $b \in \{0, -0.35, -0.5\}$ to obtain ten data points at each (n,b) pair. For all problems with b = 0.0, CPLEX found an optimal solution. For n = 60 and b = –0.5,–0.35 CPLEX was able to find optimal solutions; however, the solution time was quite long (> 1 hr). For n = 100 and b = –0.35, CPLEX could not find an optimal solution; however, the best CPLEX solution found and the best heuristic solution are within 0.2% of each other. Our analysis compares the average of the ten tests. The total cost and solution time are reported in Table 2.

Table 2: Comparison of Average Total Cost and Average Solution Time

Number of Customers (n)		60	60	60†	100	100	100†	
Facility Scale Factor (b)		–0.5	–0.35	0.0	–0.5	–0.35	0.0	Mean
% Above Best Cost	1. AD-dALA	3.25 (0)	1.91 (0)	1.19 (0)	2.66 (1)	1.59 (0)	1.75 (0)	2.06 (0)
	2. MS-dALA	0.21 (8)	0.52 (3)	2.96 (0)	0.20 (6)	0.62 (3)	2.91 (0)	1.24 (3)
	3. W1	0.84 (1)	0.10 (8)	0.01 (9)	0.64 (3)	0.08 (5)	0.03 (7)	0.28 (6)
	4. W2	0.60 (1)	0.07 (7)	0.01 (9)	0.84 (1)	0.25 (2)	0.03 (7)	0.30 (5)
	5. W1-LAt	0.65 (3)	0.30 (4)	0.02 (7)	0.52 (2)	0.19 (3)	0.07 (5)	0.29 (4)
	6. W2- Lt	0.54 (1)	0.47 (2)	0.38 (0)	0.38 (2)	0.26 (1)	0.46 (1)	0.42 (1)
	7. W2-LAt	0.85 (1)	0.53 (2)	0.43 (0)	0.31 (2)	0.32 (1)	0.60 (0)	0.51 (1)
	8. W1-Sw	0.57 (2)	0.10 (7)	0.01 (9)	0.71 (2)	0.08 (5)	0.05 (6)	0.25 (5)
	9. W2-Sw-LAt	0.59 (1)	0.51 (3)	0.43 (0)	0.38 (2)	0.28 (3)	0.64 (0)	0.47 (2)
Multiple of Best Time	1. AD-dALA	1.0 (6 s)	1.0 (6 s)	1.0 (4 s)	1.0 (15 s)	1.0 (15 s)	1.0 (10 s)	1.0 (9 s)
	2. MS-dALA	22.5	40.0	175.4	14.3	26.7	156.8	72.6
	3. W1	3.6	5.5	14.8	5.4	8.8	28.3	11.1
	4. W2	3.6	5.4	15.1	5.5	9.2	28.3	11.2
	5. W1-LAt	3.4	5.4	14.5	5.7	8.6	27.9	10.9
	6. W2- Lt	3.5	5.2	14.8	5.6	9.0	28.8	11.2
	7. W2-LAt	3.4	5.3	14.7	5.5	9.2	28.4	11.1
	8. W1-Sw	2.9	3.5	6.3	2.6	3.3	7.1	4.3
	9. W2-Sw-LAt	3.1	3.8	7.1	2.8	3.7	7.6	4.7

†Best cost obtained via CPLEX

In Table 2, the values in parentheses for the cost measurement are the number of times the best solution (within 0.05%) was found out of the ten runs. In the solution time section, the second row of data for the AD-dALA heuristic is the average solution time in seconds. To assess the statistical difference in cost between the heuristics we compare the distribution of the %AB cost measurement for the thirty total tests with $n = 100$ and $b \in \{0, -0.35, -0.5\}$ using a non-parametric Wilcoxon rank-sum test [Siegel 1994]. These results, not shown here to conserve space, are consistent with the general results observable in Table 2. Table 3 reports the evaluation measurements for each algorithm. In Table 3 we also show the allocation and construction evaluations as a combined measurement to provide a more comprehensive assessment of the total allocation evaluations required.

Table 3: Comparison of the Average Number of Evaluations using xFB Measurement

Number of Customers (*n*)		60	60	60	100	100	100	
Facility Scale Factor (*b*)		–0.5	–0.35	0.0	–0.5	–0.35	0.0	Mean
Construction Evaluations	1. AD-dALA	1.6	1.1	1.0	1.7	1.1	1.0	1.3
	2. MS-dALA	1.6	1.1	1.0	1.7	1.1	1.0	1.3
	3. W1	2.6	3.0	31.0	4.0	4.4	60.0	17.5
	4. W2	2.6	3.0	31.0	4.1	4.6	60.0	17.5
	5. W1-LAt	2.5	3.0	31.0	4.2	4.4	60.1	17.5
	6. W2- Lt	2.6	2.9	31.1	4.1	4.5	61.3	17.8
	7. W2-LAt	2.5	3.0	31.2	4.0	4.6	60.0	17.6
	8. W1-S	1.1	1.1	9.4	1.0	1.1	11.0	4.1
	9. W2-S-LAt	1.0	1.1	8.8	1.0	1.1	9.7	3.8
Location Evaluations	1. AD-dALA	1.7	1.9	3.2	1.2	1.3	1.8	1.9
	2. MS-dALA	476.7	609.6	1015.0	243.2	381.2	494.5	536.7
	3. W1	13.4	16.5	23.9	11.1	15.6	18.6	16.5
	4. W2	6.8	10.3	24.2	3.4	6.0	12.1	10.5
	5. W1-LAt	2.9	3.2	2.8	2.9	3.3	2.0	2.8
	6. W2- Lt	1.7	2.0	1.5	1.4	1.7	1.6	1.6
	7. W2-LAt	1.4	1.9	1.0	1.4	1.7	1.3	1.5
	8. W1-S	13.5	16.3	24.0	11.6	15.7	18.3	16.6
	9. W2-S-LAt	1.5	2.0	1.0	1.4	1.6	1.2	1.5
Allocation Evaluations	1. AD-dALA	1.0	1.0	1.0	1.2	1.0	1.0	1.0
	2. MS-dALA	258.9	282.3	337.6	178.8	250.1	275.2	263.8
	3. W1	6.2	8.6	14.1	4.7	7.8	13.1	9.1
	4. W2	5.7	7.9	13.7	3.9	7.3	12.3	8.5
	5. W1-LAt	2.4	4.1	8.0	1.6	2.9	7.1	4.4
	6. W2- Lt	5.5	7.1	13.8	3.8	7.0	12.6	8.3
	7. W2-LAt	2.5	3.9	7.7	1.3	3.2	7.2	4.3
	8. W1-S	6.9	8.6	14.2	5.1	7.8	12.8	9.2
	9. W2-S-LAt	2.7	3.9	7.7	1.5	3.1	7.1	4.3
Construction + Allocation Evaluations	1. AD-dALA	2.7	1.0	1.0	1.4	1.0	1.0	1.4
	2. MS-dALA	3.9	42.5	206.4	29.5	36.3	189.7	84.7
	3. W1	7.7	3.5	19.2	3.7	4.5	25.4	10.7
	4. W2	7.1	3.3	18.9	3.7	4.6	24.9	10.4
	5. W1-LAt	3.1	2.8	15.5	3.4	3.8	21.3	8.3
	6. W2- Lt	4.3	3.2	19.1	3.7	4.5	25.5	10.0
	7. W2-LAt	2.1	2.7	15.4	3.2	4.1	21.4	8.1
	8. W1-S	1.5	2.0	11.8	1.6	1.9	11.8	5.1
	9. W2-S-LAt	1.0	1.4	7.6	1.0	1.3	7.5	3.3

The following general observations from the analysis of these smaller instances will be used in the analysis of larger problems.

OBSERVATION 1: The AD-dALA procedure provides solutions within 10% of the best found solution with the fastest solution time and a relatively low number of evaluations.

This is primarily due to the computationally simple ADD/DROP method used in the construction procedure.

OBSERVATION 2: The MS-dALA procedure provides better quality solutions than the AD-dALA, but this improvement comes with a large increase in both location and allocation evaluations due to the large number of multi-starts needed to obtain a good solution. Additionally, the solutions for the MS-dALA procedure degrade as $b \to 0$ and the number of facilities in the solution increases. This result is consistent with studies of multi-start ALA procedures [Brimberg et al. 2000, Hansen et al. 1998].

OBSERVATION 3: Using the tabu search techniques we see a statistically significant increase in total cost, however, the difference is small in practical terms (on average all solutions are within 0.25% of each other using the %AB measure). This minor degradation in solution quality brings a significant reduction in location and allocation evaluations. In contrast to the MS-dALA procedure heuristics three to seven show improvement in the total cost as the number of facilities in the solution increases due to the iterative construction and improvement nature of the heuristics [Whitaker 1985].

OBSERVATION 4: By using a subset of facilities in the construction procedure we achieve a significant reduction in total allocation evaluations with only a minor degradation in the objective function (as compared to heuristics three and four). This result is consistent with variable neighborhood search studies, which show the use of multiple neighborhoods with a local improvement procedure is an effective combinatorial approach [Hansen and Mladenovic 1997].

OBSERVATION 5: The measurement of computational effort in each procedure (construction, location, and allocation) has added significantly to our understanding of the behavior of the heuristics. By using these measures we are better able to predict the behavior of the heuristics as the cost function becomes increasingly more complex.

4.4 Metaheuristic Solution Approaches for Larger Problems

Based on the observations from the study of smaller instances a subset of the heuristics are tested on larger instances with $n \in 200, 400, 600, 877$ and $b \in 0, -0.35, -0.5$. The instances are generated using the same parameters and methods as in the $n \in 60, 100$ study. Due to longer solution times only one test for each (*n,b*) pair is performed. Table 4 summarizes the heuristics analyzed. To avoid confusion, we use the same numbering of solution methods as in Table 1. Heuristic ten is a modification of heuristic nine that employs a different approach to create the subset of facility locations for use in the construction procedure. Using the same criteria used in the construction subset creation in heuristic nine, heuristic ten uses the Daskin ADD procedure used in Heuristic one to generate the construction subset facility locations.

Table 4: Heuristics Analyzed on Larger Instances

Heuristic	Facility Add/ Construction	Customer Allocation	Facility Location
1. AD-dALA	Add with no scale costs, followed by Drop procedure with scale costs	All customers	FNS using Delaunay triangulation
2. MS -dALA	AD solution provides range in *p* values to test	Same as heuristic 1	Same as heuristic 1
5. W1-LAt	Greedy Add using all unused locations	Only customers on convex hull of the current allocation	FNS checking all locations in subgraphs with new facility allocations
9. W2-S-LAt	Generate subset using simplified Whitaker ADD, then Greedy Add using unused subset locations	Same as heuristic 5	Same as model 1, but only assess subgraphs with new facility allocations
10. W2-Sa-LAt	Same as heuristic 9, but subset created using Daskin ADD procedure	Same as heuristic 9	Same as heuristic 9

4.5 Performance of Metaheuristics on Larger Problems

CPLEX continued to provide optimal solutions for all IP formulations of the problem (*b* = 0.0), with solution time for the 877 customer problem obtained in less than one minute.

Total cost and solution time for the $n \in 200, 400, 600, 877$ problems are reported in Table 5. Within the "Multiple of Best Time" section of the table, the second row of data for the AD-dALA heuristic is the average solution time in seconds. In some problems the heuristics were unable to reach a final solution due to an "out of memory" error, which we depict with "--" in the tables. Heuristics five and nine cannot solve the largest problems (n = 877). Heuristic five causes an out of memory error in the construction procedure as it attempts to add the second facility to one of the 877 facility locations. Heuristic nine causes an out of memory error in the creation of the construction. The measurements of computational effort are shown in Table 6.

Table 5: Comparison of Average Total Cost and Average Solution Time

Number of Customers (n)		**400**	**400**	**400**†	**600**	**600**	**600**†	**877**	**877**	**877**†
Facility Scale Factor (b)		**-0.5**	**-0.35**	**0.0**	**-0.5**	**-0.35**	**0.0**	**-0.5**	**-0.35**	**0.0**
% Above Best Cost	1. AD-dALA	0.22	2.80	5.24	1.22	1.60	1.76	4.23	0.00	1.79
	2. MS -dALA	0.00	0.04	3.00	0.00	0.55	2.06	0.00	1.33	2.18
	5. W1-LAt	1.13	0.00	0.20	3.33	0.00	--	--	--	--
	9. W2-Sw-LAt	0.61	0.41	0.41	0.58	0.12	2.07	--	--	--
	10. W2-Sa-LAt	0.65	0.12	0.00	0.58	0.00	0.12	4.05	0.00	0.09
Multiple of Best Time	1. AD-dALA	1.0 254s.	1.0 231s.	1.0 110s.	1.0 566s.	1.0 495s.	1.0 250s.	1.0 1366s.	1.0 1270s.	1.0 627s.
	2. MS -dALA	6.4	14.6	93.0		15.1	73.8	7.33	9.4	54.8
	5. W1-LAt	18.6	32.1	162.8	24.4	50.8	--	--	--	--
	9. W2-Sw-LAt	11.7	16.5	55.5	7.3	26.3	65.2	--	--	--
	10. W2-Sa-LAt	5.3	9.4	41.9	8.6	10.6	45.0	5.0	9.9	40.7

†Best cost obtained via CPLEX

Analyzing the results of the larger problems the following general observations can be made, some of which parallel our findings in the smaller problem study.

OBSERVATION 1: The AD-dALA procedure provides solutions within 6% of the best found solution with a relatively low number of evaluations and fast solution time. Solution quality for the AD-dALA procedure improves as $b \rightarrow 0.0$. This algorithm may be a viable alternative if a larger margin of error in the solution is acceptable and a low number of evaluations or fast solution time is required.

OBSERVATION 2: The MS-dALA procedure provides better solutions on average than the AD-dALA, but this improvement comes with a large increase in solution time and the number of evaluations required due to the large number of multi-starts needed to obtain a good solution. The solution quality of this procedure, as expected, deteriorates as the number of facilities in the final solution increases.

Table 6: Comparison of the Average Number of Evaluations using xFB Measurement

Number of Customers (*n*)		**400**	**400**	**400**	**600**	**600**	**600**	**877**	**877**	**877**
Facility Scale Factor (*b*)		**-0.5**	**-0.35**	**0.0**	**-0.5**	**-0.35**	**0.0**	**-0.5**	**-0.35**	**0.0**
Construction Evaluations	1. AD-dALA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	2. MS -dALA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	5. W1-LAt	7.6	13.9	133.6	10.0	21.8	--	--	--	--
	9. W2-Sw-LAt	1.9	3.8	31.6	3.0	5.5	50.9	--	--	--
	10. W2-Sa-LAt	1.9	3.7	31.0	2.9	3.9	33.0	4.0	3.6	25.2
Location Evaluations	1. AD-dALA	1.0	1.0	1.0	1.0	1.0	1.0	2.5	1.0	1.0
	2. MS -dALA	230.9	366.5	417.6	306.5	387.6	340.8	315.9	137.4	281.3
	5. W1-LAt	5.5	11.5	7.1	13.5	14.3	--	--	--	--
	9. W2-Sw-LAt	3.6	2.6	1.9	1.0	2.4	3.2	--	--	--
	10. W2-Sa-LAt	3.6	2.5	2.4	1.4	2.6	3.1	1.0	1.4	2.5
Allocation Evaluations	1. AD-dALA	2.4	1.2	1.0	3.2	2.0	1.0	9.8	3.0	1.0
	2. MS -dALA	459.6	285.2	385.6	806.5	511.2	332.7	1221.2	404.4	314.6
	5. W1-LAt	1.3	1.0	3.8	1.0	1.0	--	--	--	--
	9. W2-Sw-LAt	1.1	1.0	4.0	1.0	1.1	2.9	--	--	--
	10. W2-Sa-LAt	1.0	1.0	3.9	1.1	1.2	2.8	1.0	1.0	2.2
Construction + Allocation Evaluations	1. AD-dALA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	2. MS -dALA	42.9	42.9	213.2	23.7	50.0	202.0	21.3	35.8	181.6
	5. W1-LAt	11.6	11.6	62.0	9.1	17.7	--	--	--	--
	9. W2-Sw-LAt	3.3	3.3	16.3	2.7	4.5	21.8	--	--	--
	10. W2-Sa-LAt	3.2	3.2	16.0	1.8	3.2	14.7	1.5	2.8	12.0

OBSERVATION 3: The use of a subset in the construction procedure and the method to create this subset are critical factors to be able to solve large problems. Moreover, the use of a subset has a negligible impact on solution quality. This result is consistent with Ballou's [1994] customer clustering results.

OBSERVATION 4: Combining the subset in the construction procedure and tabu search techniques in the location and allocation procedures clearly provides the best overall solution cost for the range of problems tested and requires only a moderate number of evaluations. Solution quality for this procedure (heuristic 10) improved as $b \rightarrow 0.0$ due to the iterative construction and improvement nature of the heuristic. For problems with $n \geq 400$ where optimal values were found ($b = 0.0$), this algorithm was within 0.12% of the optimal value on average. Although this procedure requires a longer solution time relative to the AD-dALA heuristic, the number of evaluations to generate a solution, however, is similar for most problems suggesting that this heuristic should achieve similar solution times as the cost function becomes more complex.

5. Conclusions and Directions for Future Work

In this paper several integrative metaheuristics solution methods for large scale facility location problems that include non-linear costs that can represent risk pooling and/or production economies of scale are presented. The solution approaches are extensions of well studied methods used for the multi-source Weber, *P*-median, and uncapacitated fixed charge facility location problems. Using an empirical study a large test bed of problems was developed, that included some optimal solutions obtained using CPLEX, for comparing the solution approaches over a range of problem sizes. The metaheuristics combined ADD, DROP, ALA, variable neighborhood search, multi-start, and tabu search techniques. We showed that measuring the number of computational evaluations provided insight into the behavior of the heuristics that is not apparent when only using solution time and total cost. Specifically, these measures showed that the use of tabu search techniques in the location and allocation subroutines, and the creation of a subset of possible new facility locations in the add/construction subroutine both significantly reduced the evaluations required with minimal impact on solution quality. The advantage of each of these techniques was correlated with the number of facilities in the final solution.

There are several avenues for future research. The concepts of Tabu search can be extended in the location and allocation procedures in an attempt to further reduce the computational evaluations required. Adding VNS and/or TS to supplement and/or replace the multi-start procedure may make the discrete ALA metaheuristic more viable as these techniques have been successfully implemented in *p*-Median problems [Hansen and Mladenovic 1997, Brimberg et al. 2000]. Finally, as the metaheuristics are flexible in the type of cost function that can be solved, the models can be extended to include both strategic and tactical level decisions as well as capture other relevant cost parameters not included in typical facility location models; for example uncertainty in supply and demand, and demand correlation.

6. Acknowledgments

This research is supported, in part, by the National Science Foundation under Grant CMS-0229720 (NSF/USDOT).

7. References

1. Ballou, R.H., Rahardja, H., Sakai, N., 2002, Selected country circuity factors for road travel distance estimation, Transportation Research Part A, 36, 843-848.

2. Ballou, R.H., 1994, Measuring Transport Costing Error in Customer Aggregation for Facility Location, Transportation Journal, 33, 49-59.

3. Bischoff, M., Dachert, K., 2009, Allocation search methods for a generalized class of location–allocation problems, European Journal of Operational Research, 192, 793-807.

4. Brimberg, J.,Hansen, P., Mladenovic, N.,Taillard, E.D. ,2000, Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem, Operations Research, 48, 444–460.

5. Brimberg, J., Mladenovic, N., 1999, Degeneracy in the multi-source Weber problem, Mathematical Programming, 85, 213-220.

6. Bucci, M.J., Kay, M.G., 2006, A Modified ALA Procedure for Logistic Network Designs with Scale Economies, Industrial Engineering Research Conference, IERC 2006, Orlando Florida.

7. Bucci, M.J., Kay, M.G., Waring, D.P., 2007, A Comparison of Meta-Heuristics for Large Scale Facility Location Problems with Economies of Scale, Industrial Engineering Research Conference, IERC 2007, Nashville, Tennessee.

8. Cooper, L., 1963, Location-allocation problems, Operations Research, 11, 331–343.

9. Croxton, K.L., Zinn,W., 2005, Inventory Considerations in Network Design, Journal of Business Logistics, 26, 149-168.

10. Daganzo, C.F., 2005, Logistics systems analysis, Springer, New York, NY.

11. Daskin, M.S., Coullard, C.R., Shex, Z.M., 2002, An Inventory-Location Model: Formulation, Solution Algorithm and Computational Results, Annals of Operations Research, 110, 83-106.

12. Daskin, M.S.,1995, Network and discrete location: models, algorithms, and applications, John Wiley and Sons, New York.

13. Gendreau, M., Potvin, J.Y., 2005, Metaheuristics in combinatorial optimization, Annals of Operations Research, 140, 189-213.

14. Glover, F., Laguna, M., 1997, Tabu Search, Kluwer Academic Publishers, Boston, MA.

15. Hansen, P., Mladenovic, N., Taillard, E., 1998, Heuristic solution of the multisource Weber problem as a p-median problem, Operations Research Letters, 22, 55–62.

16. Hansen, P., Mladenovic, N., 1997, Variable Neighborhood Search for the P-median, Location Science, 5, 207–226.

17. Houck, C.R., Joines, J.A., Kay, M.G., 1996, Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems, Computers and Operations Research, 23, 587-796.

18. Humphrey, D.G., Wilson, J.R., 2000, A Revised Simplex Search Procedure for Stochastic Simulation Response Surface Optimization, Informs Journal on Computing, 12, 272-283.

19. Kay, M.G., 2006, Matlog: Logistics Engineering Toolbox, North Carolina State University (http://www.ie.ncsu.edu/kay/matlog).

20. Lowe's Companies, Inc., 2006 Annual Report, http://www.shareholder.com/lowes/annual.cfm

21. Maister, D.H., 1976, Centralization of Inventories and the 'Square Root Law,' International Journal of Physical Distribution, 6, 124-134.

22. Melachovsky, J., Prins, C., Calvo, R.W., 2005, A Metaheuristic to Solve a Location-Routing Problem with Non-Linear Costs, 11, 375–391.

23. Mirchandani, P.B., Francis, R.L., 1990, Discrete Location Theory, Wiley, New York.

24. Perl, J., Daskin, M.S., 1985, A Warehouse Location-Routing Problem, Transportation Research Part B, 19, 381–396.

25. Resende, M.G.C., Werneck, R.F., 2006, A hybrid multistart heuristic for the uncapacitated facility location problem, European Journal of Operational Research, 174, 54–68.

26. Rumelt, R.P., 2001, Note on Strategic Cost Dynamics, POL 2001-1.2, Anderson School at UCLA, pp. 2–3.

27. Siegel, A.F., 1994, Practical Business Statistics, Irwin, Burr Ridge, Ill.

28. Walgreen Company, 2006 Annual Report, http://www.shareholder.com/lowes/annual.cfm

29. Whitaker, R.A. 1985 Some Add-Drop and Drop-Add Interchange Heuristics for Non-Linear Warehouse Location, The Journal of the Operational Research Society, 36, 61–70.

30. Winston, W.L., 2003, Introduction to mathematical programming, PWS-Kent, Boston, MA.

CHAPTER 3: Modeling Inventory Pooling Effects in Facility Location

Michael J. Bucci[*], Michael G. Kay[*], Donald P. Warsing[†], Reha Uzsoy[*], James R. Wilson[*]

[*]**Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA**

[†]**Department of Business Management, North Carolina State University, Raleigh, NC 27695, USA**

Reprint of paper to be submitted IIE Transactions

Modeling Inventory Pooling Effects in Facility Location

Michael J. Bucci[*], Michael G. Kay[*], Donald P. Warsing[†], Reha Uzsoy[*], James R. Wilson[*]

[*]Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA
[†]Department of Business Management, North Carolina State University, Raleigh, NC 27695, USA

Abstract

We investigate the "Square Root Law" (SRL) and a more general concave cost function for computing inventory levels that account for pooling effects in facility location models. We compare inventory levels, total cost, solution time, and solution structure (facility locations and allocations) to solutions that use the explicit safety stock inventory calculation. For models with no correlation in demand across customer locations, we show that SRL poorly estimates the inventory required in many situations and the more general cost function always outperforms the SRL. Both approaches, however, result in underlying solution structures that are close to the solution using the explicit calculation. We then extend our analysis to models that include inter-customer correlation in demand, demonstrating the degradation of both approximation approaches as inter-customer correlation increases, and suggesting that either the explicit computation of pooled inventory levels, or better methods of approximating them, are required to generate good solutions.

Keywords

Facility location, inventory pooling, safety stock, demand correlation

1. Introduction

We investigate large-scale facility location models that account for the risk pooling effects associated with centralizing safety stocks. The models represent a single-source, single-tier distribution network that may also exhibit correlation across customer demands. The objective of the model is to locate an unknown number of distribution centers to minimize the sum of transportation and safety inventory costs. Unlike traditional location models [e.g., Daskin 1995], distribution networks have increasingly come to rely on contracts of relatively short duration in which warehouse space is rented from third parties [Armstrong and Associates 2009]. Thus, the customary fixed costs due to facility construction and infrastructure are not incurred or are significantly reduced, although the firm may incur higher variable costs.

In these environments, where the cost of safety inventory can represent a large portion of the distribution network costs, the manner in which risk pooling effects are modeled has the potential to significantly influence the nature of the network-design solution. We study the widely cited "Square Root Law" (what we will refer to as SRL_1) used to approximate such costs [Croxton and Zinn 2005] and a more general concave-cost approximation based on the warehouse size (SRL_2), and we compare the results obtained by using these approximations to an explicit computation of the safety inventory levels (SRL_0). Although there has been significant research on SRL_1 and to a lesser degree SRL_2, there has been no detailed analysis on the impact of these approximations when used in facility location models. To assess the accuracy of the use of these approximations we develop a method to compare the facility locations and demand allocations across the solutions. Due to the complexity of exact solution procedures for this form of location problem, a discrete alternate location–allocation (ALA) neighborhood search heuristic is used to obtain the solutions [Cooper 1963, Bucci et al. 2009].

Our analysis starts with an empirical study of networks containing 100 customer locations randomly selected from five-digit ZIP codes in the continental U.S., where, initially, demands at customer sites are assumed to be independent. We vary customer locations and demands, the coefficient of variation of customer demand, and the magnitude of the inventory cost relative to the transportation cost. We compare solution quality (objective function cost), solution time, and solution structure (facility locations and allocations), highlighting the impact of various problem parameter values on the accuracy of the two safety inventory approximation methods. The analysis shows that, in many cases, both SRL_1 and SRL_2 poorly estimate the inventory required and that SRL_2 always outperforms SRL_1; however, the underlying solutions with either method are not notably different from those obtained with SRL_0. We then introduce inter-customer demand correlation and conduct a more focused empirical study on the accuracy of SRL_1 and SRL_2 under these

conditions. For this study, we develop a method to create large- scale demand correlation matrices. We then show that in the presence of modest levels of correlated demands both approximation methods inadequately estimate the inventory required and result in significantly different underlying solutions. Hence, with correlation present, it appears important to explicitly compute safety inventory levels (SRL_0), or to pursue better methods to approximate them.

The remainder of the paper is organized as follows. In Section 2 we review the literature related to this problem, and we present the general formulation of the problem in Section 3. Section 4 presents the experimental design, solution approaches and computational results, and Section 5 offers conclusions and directions for future research.

2. Literature Review

We group the literature into three areas. The first group is a review of inventory pooling research, the second is location modeling that includes inventory pooling costs, and the third is literature related to correlated demands and inventory pooling.

The impacts of inventory pooling have been widely studied. Eppen [1979] studied the multi-location newsboy problem considering demand variance and demand correlation between periods, showing that inventory costs increase with the square root of the number of consolidated demands (SRL_1) if the demands at each location are uncorrelated between periods and locations and if demand variability is the same at all locations. Zinn et al. [1989] review the work of Maister [1976] and Eppen [1979], and propose a more general formulation of SRL_1, termed the Portfolio Effect, that can provide guidelines for managers considering inventory consolidation. Using a four customer problem, they show that the two assumptions underlying SRL_1 can lead to substantial error in calculating safety stock savings. They then define the Portfolio Effect, which accounts for the relative sizes of the standard deviations of demand and correlation coefficients between stocking locations.

Mahmoud [1992] expands on the work of Zinn et al. to define the Portfolio Quantity Effect (PQE), which measures the quantity reduction in safety stock due to centralization, and the Portfolio Cost Effect that includes additional potential savings from centralization beyond inventory carrying costs. He provides examples where the Portfolio Effect, which measures a percent reduction in inventory, is insufficient to find the optimal consolidation scheme. Evers and Beier [1993] investigate formulations of the problem that allocate customer demands to $m > 1$ inventory locations and show that SRL_1 and the Portfolio Effect model are not accurate for these situations. They offer a more general formula for the Portfolio Effect that relaxes the assumptions that the average lead times are equal and known with certainty.

Traditional location models such as the multi-source Weber problem [Brimberg et al. 2000], *P*-median problem [Daskin 1995], and the uncapacitated fixed charge facility location problem [Mirchandani and Francis 1990] are widely studied formulations whose objective is to locate facilities to serve a known set of demands at given customer locations to minimize transportation cost (if the number of facilities is given) or the sum of transportation costs and fixed facility costs (if the number of facilities is not specified). Exact methods and heuristics are used to solve these NP-hard problems (Daskin [1995], Hansen and Mladenovic [1997]). More recent research has extended these formulations to include inventory costs in the objective function. Nozick and Turnquist [1998] describe a method to incorporate safety stock inventory costs in a fixed-charge location model. Their work includes a detailed review of the assumptions of SRL_1 and a thorough empirical study of safety stock requirements using SRL_1. Daskin et al. [2002] and Shen et al. [2003] introduce joint location-inventory models that extend the uncapacitated fixed charge model to explicitly include the impact of demand variability on inventory pooling costs. They use a Lagrangian-relaxation based exact algorithm to solve the problem, utilizing weighting factors to adjust the relative importance of the location, transportation, and inventory costs. Snyder et al. [2007] extend this work with a model that minimizes the fixed facility, transportation, and inventory costs assuming a constant variance to mean ratio and independent customer demands.

Croxton and Zinn [2005] study a location model with inventory pooling costs using what we call SRL_1 to approximate the pooling effects. They review the assumptions of SRL_1 and its applicability to facility location problems with aggregated product information, arguing that aggregating demands for multiple products tends to result in uncorrelated demands. Vidyarthi [2007] considers a more complex multiproduct two-echelon facility location model that includes fixed facility, transportation, and inventory costs. Assuming independent demand and a piecewise linear approximation of the inventory costs, he uses a Lagrangian relaxation to obtain lower bounds and a heuristic to find feasible solutions. Ozsen et al. [2008] study a capacitated location model with risk pooling (CLMRP) that is solved using a Lagrangian algorithm. Sourirajan et al. [2008] analyze a two-stage supply chain network that includes explicit calculation of safety stock inventory costs and congestion costs at the distribution center. Using a general concave cost function (along the lines of our SRL_2) to reflect the scale economies of combined fixed and variable network costs (e.g., fixed facility costs, variable transportation costs, and variable production and/or inventory costs), Bucci et al. [2009] develop and evaluate several metaheuristic solution methods that combine algorithmic construction, allocation, and location techniques, including alternate location–allocation (ALA), variable neighborhood search (VNS), and Tabu search, to solve large-scale problems. They also determine combinations of heuristic techniques that can provide near optimal solutions with moderate computational effort. The SRL_2 cost function has also been used in earlier studies of location models by Whitaker [1985] and Melachovsky [2005] to estimate general economies of scale in warehousing.

The impacts of correlated demands on pooling costs have also been widely studied. Erkip et al. [1990] present an expression for safety stock with correlated demands between locations and between successive periods. This research assumes normally-distributed demand with equal coefficients of variation across customers, and focuses on finding the optimal ordering policy assuming a fixed allocation from inventory locations to customers. More recently, Das and Tyagi [1999] study correlated demand patterns between multiple products and offer guidelines on how to group products to minimize safety stock levels when both positive and

negative correlations are present. The inventory aggregation work of Eppen [1979] is more recently reviewed by Chopra and Meindl [2003] and covers the impact of aggregation on safety inventory for situations with independent demand and correlated demand. Using the notation of Chopra and Meindl, the aggregated safety stock is

$$I = F^{-1}\ CSL\quad \sigma_L^F, \tag{3.1}$$

where I is the safety stock required for a given facility, F^{-1} is the inverse cumulative distribution function (cdf) of demand allocated to this facility over the lead time to replenish inventory at this facility, *CSL* is the cycle service level (i.e., the expected in-stock probability level across inventory cycles), and σ_L^F is the standard deviation of demand during the replenishment lead time at the facility. Furthermore, Chopra and Meindl show that

$$\sigma_L^F = \sqrt{L}\sqrt{\sum_{i=1}^{k}\sigma_i^2 + 2\sum_{i<j}\rho_{ij}\sigma_i\sigma_j}, \tag{3.2}$$

where L is the supplier lead time, σ_i and σ_j are the standard deviations of demand at customers i and j, respectively, and ρ_{ij} is the correlation coefficient of the demands of customers i and j.

Xu and Evers [2003] review inventory pooling effects with correlated demands and specifically address the challenge and criticality of creating valid correlation matrices. They offer methods to generate small correlation matrices (up to a 3-by-3 matrix) and methods to assess the validity of a given correlation matrix. Corbett and Rajaram [2006] describe a copula method to generate correlation values between a pair of demands for multivariate normal product demands as well as several non-normal distributions. Dorey and Joubert [2005] also review the difficulty of generating valid correlation matrices and discuss the use

of the copula to generate correlation values between a pair of values. What is lacking in these papers, however, is a method to generate correlation values between more than a few demand values.

Our research tries to combine these three streams of literature into a single model to study the impacts of inventory pooling on solutions to facility location problems with uncorrelated or correlated demands. In order to perform this analysis we developed a method to generate large-scale correlation matrices for customer demands. We then assess the suitability of the SRL_1 and SRL_2 approximations, from a cost and solution structure perspective, for a range of parameter settings showing they do not perform well in many situations. We present the model formulation in the next section.

3. Model Formulation

Given *n* uncapacitated distribution facilities that serve *m* customer locations, our formulation of the inventory-location problem (ILP) is as follows:

ILP:
$$\min z = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} X_{ij} + K \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{m} X_{ij} \sigma_j^2 + 2 \sum_{j<y} X_{ij} X_{iy} \rho_{jy} \sigma_j \sigma_y} \tag{3.3}$$

subject to

$$\sum_{i=1}^{n} X_{ij} = 1 \text{ for } j = 1, 2, \ldots, m \tag{3.4}$$

$$X_{ij} \in \{0,1\} \quad \text{for } i = 1, 2, \ldots, n \;\; j = 1, 2, \ldots, m \tag{3.5}$$

$$X_{iy} \in \{0,1\} \quad \text{for } i = 1, 2, \ldots, n \;\; y = 1, 2, \ldots, m, \tag{3.6}$$

In ILP, C_{ij} reflects the total transportation cost to move goods from facility *i* to cover the annual demand at customer *j*. This transportation cost is modeled as a one-way full truckload shipment cost and is calculated as the product of the freight transport charge ($/mi), the estimated road distance (mi) between the customer and facility locations, and the total annual demand (truckloads) at the customer. *K* is a specified constant that combines the annual cost of holding a unit in inventory at any facility in the network, the constant and common replenishment lead time across all facilities, and the common *CSL* across all facilities. The decision variables in ILP are the binary integer variables X_{ij}, which take a value of 1 if customer demand at *j* is supplied by facility *i* and set to 0 otherwise. Thus, the first term in the objective function of ILP, computes the total annual cost of transporting goods from the facilities to the customers. The second term computes the total safety inventory cost for all facilities, following the formulation of Chopra and Meindl [2003] as given in expressions (3.1) and (3.2). Constraints (3.4) require that each customer is served by only one facility. Constraints (3.5) and (3.6) are integer constraints. To evaluate SRL_1 we replace the second term in the objective function of ILP with

$$I_1^{(n)} = I^P n^{0.5}, \tag{3.7}$$

where

$$I^P = I_{0,\rho=0} = K\sqrt{\sum_{x=1}^{m} \sigma_x^2}, \tag{3.8}$$

Expression (3.7) is a rearrangement of the "Square Root Law," with I^P denoting the inventory cost for the single-facility solution assuming inter-customer demands are uncorrelated. To assess the accuracy of our general concave cost function SRL_2 we replace the second term in the objective function of ILP with

$$I_2^{(n)} = I^P \sum_{i=1}^{n} f_i^{0.5}, \tag{3.9}$$

where f_i represents the fraction of the total demand allocated to facility *i*. Again, the single facility solution I^P is the basis for calculating the inventory cost as facilities are added to the solution.

4. Computational Results

4.1 Experimental Design

We model a single-source single-tier distribution network with a single product that may have inter-customer correlated demands. The customer locations and the potential facility locations are the five-digit ZIP codes in the continental U.S. with non-zero population that are located between –80 and –90 degrees longitude (9,744 in total). In generating the costs for the models we assume the supply chain mirrors existing retail networks in the U.S., which typically contain between five and twenty distribution facilities to serve the majority of the continental U.S. For example, Lowe's [2006] and Walgreens [2006] respectively utilize 11 and 13 regional distribution centers to serve the continental U.S. The great circle distance in statute miles, magnified by a circuity factor of 1.2 to approximate actual travel distances, is used to determine the distance between facilities and customer locations [Ballou et al. 2002].

Let $N = 1, 2, \ldots, 9744$ be the (arbitrarily) ordered set of all 9744 customer locations. For each problem instance that is formulated and solved, we define $N_n \subseteq N$ ($|N_n| = n$) to be the customer location set for a problem, with $n = 100$. Further, let $M \subseteq N_n$ be the set of *m* candidate facility locations (i.e., $|M| = m$) for a particular problem instance.

Our first set of experiments assumes independent inter-customer demands (i.e., $\rho = 0$) within a full factorial experiment with four parameters. The work of Rardin and Uzsoy [2001] was used as a guide for the experimental design and analysis. The first experimental

parameter is the sampling method used to select the customer locations: we sample from the customer set with either (1) a random population-weighted selection without replacement or, (2) make a random selection with equal probability without replacement. This provides two different sets of customer locations, with the population-weighted selection having more clustered demand points than the more uniformly distributed random selection. The second parameter, customer weights, has two levels: population-weighted demands or equal demand weights. The intent of including this factor is to assess the impact of demand concentration on the solutions. In both cases, the customer weights are normalized to maintain a total demand of 10,000 units per year. The third parameter is the inventory cost weighting factor β which has two levels and is used to adjust the inventory cost relative to the transportation cost. The values for β were set, after some initial experimentation, to obtain solutions with the desired range of facilities (5–20). The final parameter is the coefficient of variation (CV) of demand which has three levels:
$cv_i = 0.2, cv_i = 0.35, cv_i \sim U\ \ 0.1, 0.35$ for each facility *i*. In lieu of real data we chose these fixed values based on our assumption that these would be low to moderate demand variations seen by a typical a distribution center. The stochastic case was studied to assess the impact of a non-constant CV value.

The first goal for this set of experiments was to compare the inventory cost estimates from the two approximations (SRL_1 and SRL_2) to the results of the exact expression (SRL_0) for a given set of facility locations and allocations. To achieve this, we solved each problem instance using SRL_0 to compute inventory cost and then, without altering the location or allocation decisions, recalculated the inventory cost using SRL_1 and SRL_2. The second goal was to compare the total cost and the underlying solution structure for each problem when solved from the outset with the objective function calculated using the different safety inventory formulas. For the comparison of total cost, the inventory cost for each solution was recalculated using SRL_0 to allow for an "apples to apples" comparison which also provides some insight into the similarity of the solutions in terms of specific locations and allocations. We call this the "Adjusted Total Cost."

To compare solution structures we also created two measures to assess the relative locations of the facilities using SRL_1 and SRL_2 versus the SRL_0 solution. The first measure "% of common facility locations: exact comparison" shows the percentage of facilities that are in exactly the same location as the facilities in the SRL_0 solution. In some solutions the number of facilities in the solutions may not be identical, and in these cases we use the smaller value for the denominator in the measurement. The second measure, "% of common facility locations– proximity comparison," shows the percentage of facilities that are within a 30-mile radius of a facility in the SRL_0 solution. This distance was determined to be reasonable as qualitative considerations could reasonable justify shifting the final location of a given facility in the computed solution by this amount. Moreover, since the solution space is relatively large (roughly 500 miles by 900 miles in size) this distance reasonably represents a "zero-distance" difference in the solutions. We introduced this second measure after comparing solutions and observing several facilities not in exactly the same location, but relatively close to one another.

In the second set of experiments inter-customer demand correlation is incorporated into the model to assess the impact of correlation on the solution cost and structure. The first requirement for this analysis, using the insights of Xu and Evers [2003], was to create a valid (i.e. positive semi-definite) correlation matrix, which is a non-trivial task even for small problems. Lacking actual demand data and guidance from the literature on data generation techniques for correlation matrices of larger size, we developed a method to determine correlation values based on the assumption that as distance between a customer pair increases the level of correlation between the customer pair decreases. For example, two customers located in Pennsylvania are likely to have more highly correlated demands for a given item than a customer in Pennsylvania and a customer in Arizona due to their common geography (e.g., common weather) and the fact that the consumers that ultimately drive their respective demands for the item are likely to display more consistent buying behavior (e.g., tastes and preferences). As a first approach, we limit the possible correlation values for each pair of customer sites to three values by geographically partitioning the

solution space into three customer groups based on their location. An arbitrary set of three latitude values were chosen to create the three groups as shown in Figure 1. The correlation coefficients are then determined using the following methodology. If the pair of customer locations (*i*,*j*) are located in the same group (i.e. *AA*, *BB*, or *CC*) they have a common correlation value ρ_0. If the customer locations (*i*,*j*) are located in adjacent groups (i.e. *AB* or *BC*) they are assigned a common correlation value ρ_1. If the customer locations (*i*,*j*) are in distant groups (i.e., *AC*) they have a common correlation value ρ_2. In general we assume that the correlation ρ_{ij} between the demands of two customers, one in group *i* and the other in group *j*, has the form $\rho_{ij} = \rho_{|i-j|}$. For a situation in which there are three groups, the relationship between the correlation coefficients ρ_0, ρ_1, and ρ_2 is detailed in Appendix 3A.

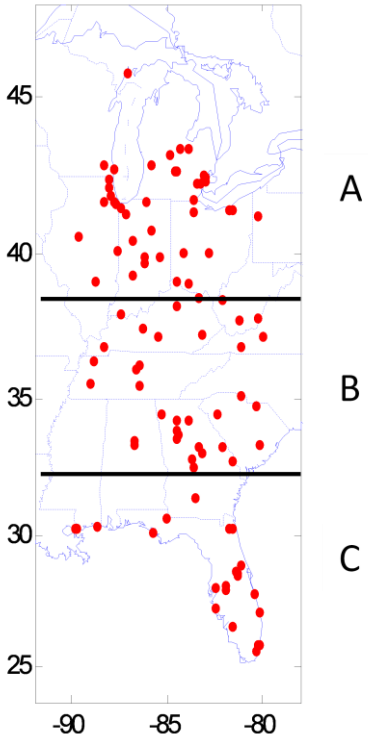

Figure 1: Partitioning of the Solution Space for the Creation of the Correlation Matrix

4.2 Metaheuristic Solution Approach

Given the computational burden of exact procedures, especially for location models incorporating the explicit inventory calculation, we opt for the "W2-S-Lat" heuristic described in Bucci et al. [2009] that has been shown to consistently produce solutions within 1% of the optimal total cost in modest CPU times (approximately 1300 seconds to solve a 400-customer problem with 400 potential facility locations using a 3.2 GHz Intel Pentium 4 computer with 1527 MB of memory running Windows XP). As our primary motivation is the impact of the different inventory modeling approaches on the solution cost and structure, obtaining an optimal solution is not critical for this analysis. The general structure of the heuristic is a combination of a constructive ADD procedure followed by a discrete ALA improvement procedure as shown in Figure 2 [Daskin 1995, Whitaker 1985]. We solve thirty instances (i.e., 30 different samples of 100 customer locations) at each experimental setting, and each inventory cost expression is tested on each instance. We compare solution cost, solution structure, and solution time, by computing the average value across the thirty replications. The problems were generated and the heuristic solution procedures were coded and executed in MATLAB 7.2, utilizing the Matlog toolbox [Kay 2006] as a building block for the models. All tests were run on a 3.2 GHz Intel Pentium 4 computer with 1527 MB of memory running Windows XP.

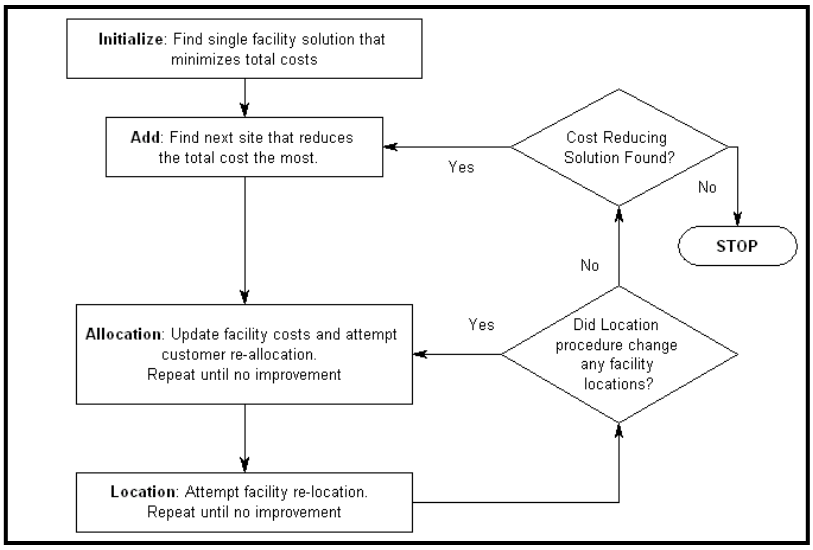

Figure 2: General Solution Procedure for W2-S-Lat Heuristic (Bucci et al. 2009)

4.3. Results from Model without Correlation

In the following three subsections we analyze the accuracy of the two inventory cost approximation methods, the impact of the experimental settings on the results, and the total cost and solution structure for the experiments with no inter-customer correlation.

4.3.1 Inventory Level Comparison

For each of the twenty-four experiments we found the solution using SRL_0. We then recalculated the estimated inventory level for the same location-allocation solution using SRL_1 and SRL_2 and compared the differences in inventory across the three techniques. As seen in Table 1 and Figure 3, SRL_1 always overestimates the inventory cost (by 7.1% on average), while SRL_2 does so to a lesser extent (by 2.5% on average). The overestimation of the inventory cost in SRL_1 is due to its assumption that demand will be uniformly allocated among all open facilities. Nozick and Turnquist [1998] characterize this as the "square-roof effect" in that SRL_1 always provides the highest possible safety stock inventory scenario. Figure 4 displays the average size for the eleven largest facilities in each solution for a

subset of the solutions and Table 2 displays the average number of facilities and the average size of the five largest facilities in each experiment, with the data sorted by the average number of facilities in the solution. If the equal allocation assumption were valid, the rows in the table would have equal facility size values, resulting in horizontal lines in Figure 4. Clearly, the "equal allocation assumption" is not accurate for the range of instances tested.

The overestimation found in SRL_1 is also apparent with SRL_2 but to a lesser extent since SRL_2 explicitly accounts for facility size. The following proposition shows that SRL_2 is equal to SRL_0 when all customer demands are equal with a common CV.

Proposition: If all customers have independent demands, a common coefficient of variation (*c*), and a common demand weight, then the total safety stock inventory levels across the network are the same for both SRL_0 and SRL_2.

Proof: For SRL_0, the explicit computation of the inventory at a single facility *x* is $I_{x,0} = K\sqrt{\sum_{i=1}^{m_x}\sigma_i^2}$, where m_x is the number of customers allocated to facility *x* from the complete customer set *M*. (Recall also that the "count" of *M* is $|M| = m$.) The common CV and common demand weight imply that $\sigma^2 = (c\mu)^2$ for all customers, and therefore, the inventory at facility *x* can then be stated as $I_{x,0} = Kc\mu\sqrt{m_x}$. From expression (3.9) for SRL_2, the inventory at a single facility *x* is $I_{x,2} = I^{(1)} f_x^{0.5} = \left(K\sqrt{\sum_{i=1}^{m}\sigma_i^2} \right) f_x^{0.5}$, where $f_x = m_x/m$ is the fraction of the total demand allocated to facility *x*. Again, with $\sigma^2 = (c\mu)^2$, we obtain $I_{x,2} = Kc\mu\sqrt{m}\cdot\sqrt{m_x/m} = Kc\mu\sqrt{m_x}$. Since the inventory at any single facility *x* is the same for both procedures, it follows that the system-wide inventories are equal—i.e., that $I^{(n)} = \sum_{x=1}^{n} I_{x,0} = \sum_{x=1}^{n} I_{x,2}$, proving the result.

Table 1: Inventory Cost Analysis for Independent Demand Case

Experiment Code	Customer location selection	Customer Weights	β	CV	Avg. # of Facilities	SRL_2 avg. % above SRL_0	SRL_1 avg. % above SRL_0
WWHL	Wt. rand.	Wt.	0.4	0.2	8.2	3.7%	9.7%
WWHH	Wt. rand.	Wt.	0.4	0.35	4.6	1.9%	4.3%
WWHR	Wt. rand.	Wt.	0.4	Random	6.1	3.2%	7.7%
WWLL	Wt. rand.	Wt.	0.3	0.2	10.9	4.7%	12.3%
WWLH	Wt. rand.	Wt.	0.3	0.35	6.1	2.8%	6.8%
WWLR	Wt. rand.	Wt.	0.3	Random	8.1	4.2%	10.4%
WEHL	Wt. rand.	Equal	0.4	0.2	7.8	0.0%	3.2%
WEHH	Wt. rand.	Equal	0.4	0.35	4.2	0.0%	2.4%
WEHR	Wt. rand.	Equal	0.4	Rand.	5.4	0.2%	2.3%
WELL	Wt. rand.	Equal	0.3	0.2	10.3	0.0%	4.2%
WELH	Wt. rand.	Equal	0.3	0.35	5.6	0.0%	2.2%
WELR	Wt. rand.	Equal	0.3	Rand.	7.5	0.3%	3.5%
RWHL	Rand.	Wt.	0.4	0.2	10.5	7.1%	14.8%
RWHH	Rand.	Wt.	0.4	0.35	5.5	3.6%	8.8%
RWHR	Rand.	Wt.	0.4	Rand.	7.1	5.6%	12.6%
RWLL	Rand.	Wt.	0.3	0.2	13.5	8.4%	16.0%
RWLH	Rand.	Wt.	0.3	0.35	7.7	5.7%	12.9%
RWLR	Rand.	Wt.	0.3	Rand.	10.4	7.6%	15.4%
REHL	Rand.	Equal	0.4	0.2	8.9	0.0%	3.2%
REHH	Rand.	Equal	0.4	0.35	5.4	0.0%	3.5%
REHR	Rand.	Equal	0.4	Rand.	6.4	0.3%	3.7%
RELL	Rand.	Equal	0.3	0.2	12.3	0.0%	4.1%
RELH	Rand.	Equal	0.3	0.35	5.6	0.0%	2.2%
RELR	Rand.	Equal	0.3	Rand.	8.6	0.3%	3.5%

Experiment Code legend
First character, Customer Location Selection : **W** = Population weighted, **R** = Random selection
Second character, Customer Weights: **W** = Population weighted, **E** = Equal weights
Third character, β: **H** = 0.4, **L** = 0.3
Fourth character, CV: **L** = 0.2, **H** = 0.35, **R** = random in range of 0.1-0.35

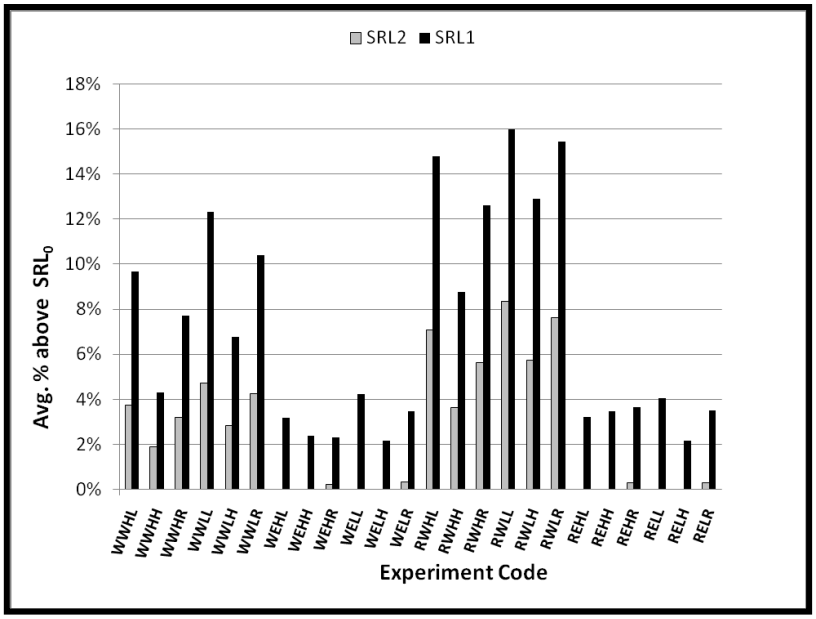

Figure 3: Inventory Difference between SRL_0 and SRL_1 and SRL_2 for all twenty four experiments

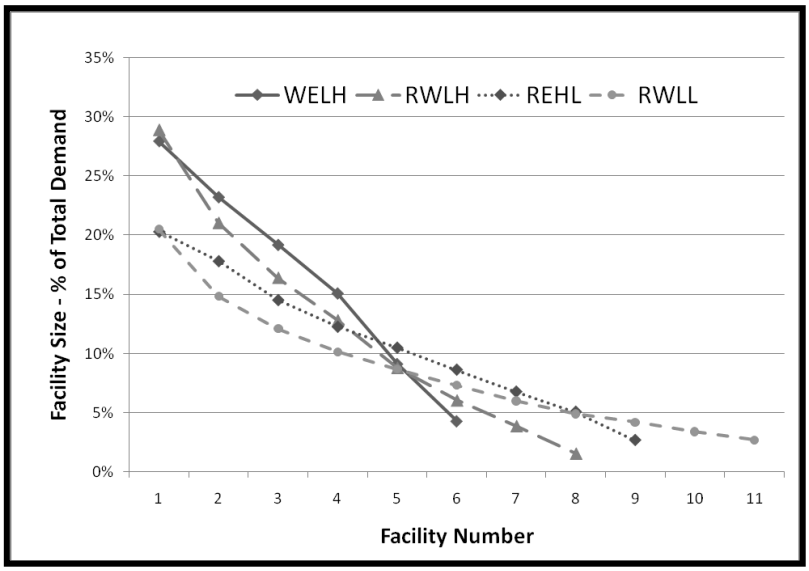

Figure 4: Average Size of Facilities for a subset of the Experiments (facilities sizes sorted largest to smallest)

Table 2: Average Facility Size for the Five Largest Facilities in each Experiment

		Avg. % of total demand for five largest facilities				
Problem	Avg. # of Facilities	1	2	3	4	5
WEHH	4.2	37%	28%	20%	12%	3%
WWHH	4.6	35%	26%	19%	14%	4%
WEHR	5.4	29%	24%	19%	15%	8%
REHH	5.4	33%	25%	19%	13%	8%
RWHH	5.5	34%	24%	19%	13%	7%
WELH	5.6	28%	23%	19%	15%	9%
RELH	5.6	28%	23%	19%	15%	9%
WWHR	6.1	32%	22%	17%	14%	8%
WWLH	6.1	32%	22%	18%	15%	9%
REHR	6.4	28%	22%	18%	14%	10%
RWHR	7.1	30%	22%	17%	13%	9%
WELR	7.5	24%	20%	17%	14%	10%
RWLH	7.7	29%	21%	16%	13%	9%
WEHL	7.8	23%	20%	16%	13%	10%
WWLR	8.1	30%	18%	15%	12%	9%
WWHL	8.2	30%	19%	14%	11%	10%
RELR	8.6	21%	18%	15%	13%	11%
REHL	8.9	20%	18%	15%	12%	11%
WELL	10.3	21%	16%	14%	11%	9%
RWLR	10.4	24%	18%	14%	12%	9%
RWHL	10.5	24%	17%	14%	12%	9%
WWLL	10.9	28%	16%	12%	10%	9%
RELL	12.3	17%	14%	12%	10%	9%
RWLL	13.5	20%	15%	12%	10%	9%

4.3.2 Parameter Impact on Results

Table 3 groups the data in Figure 3 to show the impact of the parameter settings on the accuracy of SRL_1 and SRL_2. Generally, the deviation of both approximations from the result of the explicit model increases with a larger number of facilities in the final solution and a higher variation in the size of the facilities within the solution. The increase in error with variation in the facility sizes is directly due to the inaccuracy of the "equal allocation assumption." The larger number of facilities amplifies this effect as the error from the "equal allocation assumption" is compounded over more facilities. Relative to the other parameters the weighting of customer demand causes both a large variation in facility sizes and a larger number of facilities in the solution, resulting in the largest error of the approximation techniques.

Table 3: Impact of Parameter Values on the accuracy of SRL_1 and SRL_2

	Solution Data				
Parameters	Avg. # of facilities	Std. dev. of facility capacities	SRL_2 avg. % above SRL_0	SRL_1 avg. % above SRL_0	Explanation
Customer Location Selection					Random locations create less clustering, resulting in a larger number of facilities but lower variation in facility sizes
Wt. Rand.	7.1	8.8%	1.8%	5.8%	
Rand.	8.5	7.7%	3.2%	8.4%	
Customer Weights					Weighted customer demands cause both larger number of facilities and higher variation in facility sizes
Wt.	8.2	8.7%	4.9%	11.0%	
Equal	7.3	7.8%	0.1%	3.2%	
β					Decreasing the inventory costs relative to the transportation cost results in a larger number of facilities but lower variation in facility sizes
0.4	6.7	9.1%	2.1%	6.3%	
0.3	8.9	7.4%	2.8%	7.8%	
CV					Increasing CV results in a larger number of facilities but lower variation in facility sizes
0.2	10.3	6.4%	3.0%	8.4%	
0.35	5.6	10.2%	1.8%	5.4%	
Rand. (0.1-0.35)	7.4	8.1%	2.7%	7.4%	

4.3.3 Total Costs, Solution Time, and Solution Structure Comparison

We now solve a subset of the twenty-four instances using each of the three inventory formulas from the outset, and then compare the total cost and the solution structure of the solutions (see Table 4). The total cost was analyzed using a "percent above best" measurement that has been used previously in facility location research [Brimberg et al. 2000]. For the total costs comparison we also recalculate the total cost for each final solution using SRL_0 to allow for a common baseline comparison of the total costs ("Adjusted Total Cost," as described earlier). The Adjusted Total Cost is also used to assess this two-step sequential approach as a solution technique. Paralleling the inventory cost analysis, SRL_2 always outperforms SRL_1 and the largest error in the inventory analysis (exp. RWLL) also provides the largest error in the solution cost results. The improvement in the percentage difference from SRL_0 for Adjusted Total Costs as compared to Total Costs seems to justify the use of the secondary step of using SRL_0 to compute the Adjusted Total Cost from the SRL_1 or SRL_2 solution. The facility location comparison follows these results as an increase in common locations results in a better approximation of the SRL_0 inventory costs across the network. Overall, the cost differences using Adjusted Total Cost are relatively small, so the use of either SRL_1 or SRL_2 as the objective function in the heuristic used to obtain the solutions, followed by SRL_0 for a final cost estimate is a practical approach for these types of problems.

Table 4: Average Total Cost and Solution Structure Values

					Total Cost, Adjusted Total Cost: % above best		% of common facility locations: exact comparison		% of common facility locations: proximity comparison	
exp	location	demand	β	c.v.	SRL_2	SRL_1	SRL_2	SRL_1	SRL_2	SRL_1
WWHL	wt. rnd.	wt	0.4	0.2	1.0, 0.2	1.9, 1.1	84.2	70.7	89.7	82.9
WWLR	wt. rnd.	wt	0.3	rand	3.4, 0.4	4.2, 1.4	78.5	63.4	87.2	80.4
WELL	wt. rnd.	equal	0.3	0.2	1.0, 0.1	1.9, 0.4	100.0	74.3	100.0	87.0
WELR	wt. rnd.	equal	0.3	rand	0.3, 0.3	1.2, 0.4	76.8	69.5	82.8	81.1
RWLL	rand	wt	0.3	0.2	4.3, 1.5	6.5, 2.2	87.6	81.9	92.1	87.7
RWLR	rand	wt	0.3	rand	3.1, 1.1	4.7, 1.9	83.8	76.2	89.0	82.8
RELL	rand	equal	0.3	0.2	0.1, 0.0	1.3, 0.3	100.0	65.6	100.0	77.1

4.4 Results from Models with Correlation

Based on the independent demand results we use experiment WWHL as a baseline to assess the impact of low and moderate inter-customer demand correlation on the inventory cost, total cost, and solution structure when using the accuracy of the two safety inventory approximation methods. The low correlation values are $\rho_{ij} \in 0.1, 0.05, 0.0$ and the moderate correlation values are $\rho_{ij} \in 0.3, 0.1, 0.0$.

4.4.1 Inventory Level Comparison

Figure 5 shows the inventory levels required when a single instance for the problem was solved with the explicit inventory calculation (SRL_0) for the no correlation, low correlation, and moderate correlation cases as well as the inventory levels required with SRL_1 and SRL_2. As correlation increases a significant increase in inventory is required and the curve becomes flatter due to the reduced pooling benefits. (We note that if demands were perfectly positively correlated, i.e., $\rho_{ij} = 1 \ \forall \ i, j$, the inventory required would be 155 units.) We also observe the relatively large error in inventory cost when correlation is introduced versus the error in the no correlation case.

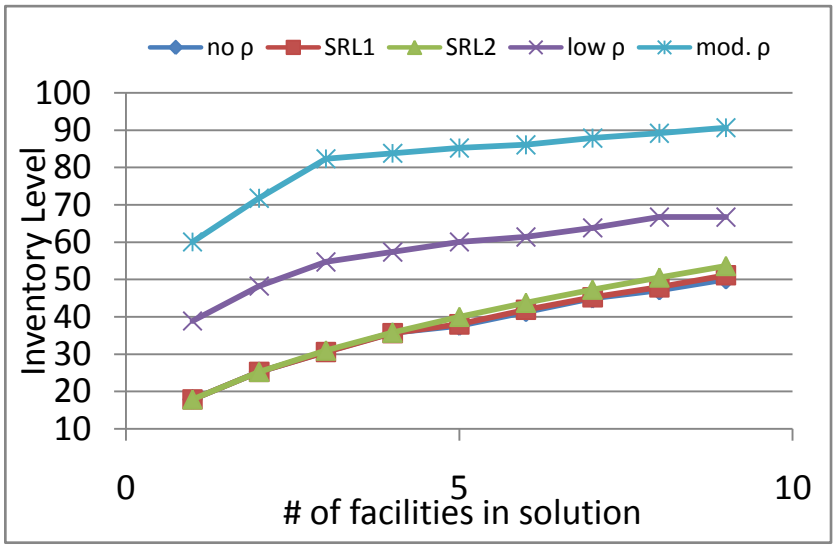

Figure 5: Inventory Level Requirements as the Number of Facilities in the Solution Increases

4.4.2 Total Cost, Solution Time, and Solution Structure Comparison

Using experiment WWHL and thirty experiment instances we solve the two correlated cases using each of the three inventory formulas from the outset and then compare the total cost, solution time, and the solution structure. As in the earlier study, for the total costs comparison we also calculate the total cost for each final solution using SRL_0 (the "Adjusted

Total Cost"). Solution times are analyzed as a "multiple from best" measurement that is calculated by taking the heuristic value and dividing it by the best value found among all the heuristics. Table 5 shows the results of this analysis. As in Figure 5, the inventory cost error increases dramatically when correlation is present. Likewise, as correlation is introduced it leads to a larger error in the Total Costs and Adjusted Total Costs. The variation in the number of facilities in the final solution also increases with inter-customer correlation as does the error in the common facility metric. The improvement in cost when using SRL_0 versus SRL_1 does come with an increase in solution time.

Table 5: Average values from Inter-customer Correlation experiments

		Inter-customer Demand Correlation		
		Zero	Low	Moderate
Number of facilities in the final solution	SRL_0	8.2	10.8	14.2
	SRL_2	7.8	7.8	7.8
	SRL_1	6.6	6.6	6.6
Inventory Cost: % from SRL_0	SRL_2	3.7	-19.3	-31.8
	SRL_1	9.7	-13.9	-26.9
Total Cost, Adjusted Total Cost : % above best	SRL_2	1.9, 0.2	–16.7, 1.6	–32.5, 4.1
	SRL_1	3.9, 1.1	–15.0, 2.8	–31.1, 5.4
% of facility locations that are common: exact comparison	SRL_2	84	78	60
	SRL_1	71	63	50
% of facility locations that are common: proximity comparison	SRL_2	90	87	80
	SRL_1	83	80	78
Solution time** : multiple from best	SRL_0	3.5	15.6	27.9
	SRL_2	1.7	2.2	2.6
	SRL_1	1	1	1

** Solution times are compared by solving the experiment for a common number of facilities in the final solution.

5. Conclusions and Directions for Future Work

We investigate three approaches to modeling inventory pooling costs in facility location models. In models with no inter-customer correlation of demand, the "Square Root Law"

(SRL_1) was shown be fairly inaccurate at estimating inventory costs primarily due to the equal allocation assumption. However, the underlying solution structure provided by SRL_1 was reasonably close to that found using the explicit inventory formula SRL_0, suggesting that SRL_1 could be used for these types of problems if the cost for the final solution was recalculated using SRL_0. A more general concave cost function, SRL_2, which accounts for facility size more explicitly, was shown to always outperform SRL_1 with minimal additional computational effort. To assess models with inter-customer correlation a simple methodology was developed to produce a valid correlation matrix under the premise that correlation between a customer pair decreases as the distance between a customer pair increases. In models with low and moderate inter-customer correlation of demand, both SRL_1 and SRL_2 poorly estimate the inventory required and have significantly different underlying solution structures relative to SRL_0.

Further research should explore different ways to incorporate the explicit calculation or new approximations, such as changing the exponent in both SRL_1 and SRL_2, into heuristic solutions. It would be useful to extend the models to assess the impact of additional costs such as cycle inventory cost and vehicle routing considerations. The development of a practical methodology to generate large inter-customer correlation matrices with many potential correlation coefficient values would also be helpful for the further study of models with correlated customer demands.

6. Appendix 3A: Procedure for Generating Valid Correlation Matrices[1]

To characterize the feasible combinations of ρ_0, ρ_1, and ρ_2, we need some additional notation. Let m_i denote the number of customers in group i, with $m_i \geq 2$ for $i = 1, 2, 3$;

[1] I am indebted to Dr. James R. Wilson, of the Fitts Department of Industrial & Systems Engineering, North Carolina State University, for his substantial contributions to this procedure and proof. The procedure was motivated by my proposed, geographically-based structure for generating valid correlation matrices, but Dr. Wilson is to be credited with formalizing that structure and proving its validity

and let $\mathbf{I}_{m_i}$ denote the $m_i \times m_i$ identity matrix for $i = 1,\ 2,\ 3$. Similarly we let $\mathbf{1}_{m_i}$ and $\mathbf{0}_{m_i}$ denote the m_i-dimensional column vectors of ones and zeros, respectively; and we let $\mathbf{U}_{m_i} = \mathbf{1}_{m_i}\mathbf{1}'_{m_i}$ denote the $m_i \times m_i$ matrix of ones for $i = 1,\ 2,\ 3$. We let $\mathbf{0}_{m_i, m_j} = \mathbf{0}_{m_i}\mathbf{0}'_{m_j}$ denote the $m_i \times m_j$ matrix of zeros for $i,\ j = 1, 2, 3$. To simplify some of the notation, we let $m = m_1 + m_2 + m_3$ denote the total number of customers whose demands are to be modeled; and because $m_i \geq 2$ for $i = 1, 2, 3,$ we always have $m \geq 6$. Finally for $i = 1, 2, 3,$ we let $\mathbf{Q}_i$ denote an $m_i \times (m_i - 1)$ matrix such that $\left[m_i^{-1/2}\mathbf{1}_{m_i} \quad \mathbf{Q}_i\right]$ is an $m_i \times m_i$ orthogonal matrix. In particular, note that for $i = 1, 2, 3,$ we can always start from the vector $m_i^{-1/2}\mathbf{1}_{m_i}$ in m_i-dimensional Euclidean space $\Re^{m_i}$ to select a set of $m_i \times 1$ vectors $\left\{m_i^{-1/2}\mathbf{1}_{m_i}; \mathbf{V}_j : j = 1, \ldots, m_i - 1\right\}$ that form a orthonormal basis of $\Re^{m_i}$; and thus we can take $\mathbf{Q}_i = \left[\mathbf{V}_1, \ldots, \mathbf{V}_{m_i - 1}\right]$. For a specific example of such a construction, see p. 71 of Searle [1982].

Proposition 3A-1. The $m \times m$ *correlation matrix* ***R*** *of customer demands has the form*

$$\begin{bmatrix} (1-\rho_0)\mathbf{I}_{m_1} + \rho_0\mathbf{U}_{m_1} & \rho_1\mathbf{1}_{m_1}\mathbf{1}'_{m_2} & \rho_2\mathbf{1}_{m_1}\mathbf{1}'_{m_3} \\ \rho_1\mathbf{1}_{m_2}\mathbf{1}'_{m_1} & (1-\rho_0)\mathbf{I}_{m_2} + \rho_0\mathbf{U}_{m_2} & \rho_1\mathbf{1}_{m_2}\mathbf{1}'_{m_3} \\ \rho_2\mathbf{1}_{m_3}\mathbf{1}'_{m_1} & \rho_1\mathbf{1}_{m_3}\mathbf{1}'_{m_2} & (1-\rho_0)\mathbf{I}_{m_3} + \rho_0\mathbf{U}_{m_3} \end{bmatrix};$$

and ***R*** *is positive definite if and only if the following conditions hold simultaneously:*

$$-\frac{1}{m_1 - 1} < \rho_0 < 1, \tag{3A.1}$$

$$|\rho_1| < \sqrt{\frac{[1 + (m_1 - 1)\rho_0][1 + (m_2 - 1)\rho_0]}{m_1 m_2}}; \tag{3A.2}$$

and given the values of ρ_0 *and* ρ_1 *satisfying (3A.1) and (3A.2), we take*

$$\rho_2^* < \rho_2 < \rho_2^{**}, \tag{3A.3}$$

where $\rho_2^* < \rho_2^{**}$ *and* ρ_2^*, ρ_2^{**} *are the roots of the following quadratic equation in* ρ_2:

$$\det(\mathbf{B}) = 0, \tag{3A.4}$$

with

$$\mathbf{B} \equiv \begin{bmatrix} 1+(m_1-1)\rho_0 & \rho_1\sqrt{m_1 m_2} & \rho_2\sqrt{m_1 m_3} \\ \rho_1\sqrt{m_1 m_2} & 1+(m_2-1)\rho_0 & \rho_1\sqrt{m_2 m_3} \\ \rho_2\sqrt{m_1 m_3} & \rho_1\sqrt{m_2 m_3} & 1+(m_3-1)\rho_0 \end{bmatrix}. \tag{3A.5}$$

Proof. It is easy to check that the $m \times m$ matrix

$$\mathbf{H} = \begin{bmatrix} m_1^{-1/2}\mathbf{1}'_{m_1} & \mathbf{0}'_{m_2} & \mathbf{0}'_{m_3} \\ \mathbf{0}'_{m_1} & m_2^{-1/2}\mathbf{1}'_{m_2} & \mathbf{0}'_{m_3} \\ \mathbf{0}'_{m_1} & \mathbf{0}'_{m_2} & m_3^{-1/2}\mathbf{1}'_{m_3} \\ \mathbf{Q}'_1 & \mathbf{0}_{m_1-1,m_2} & \mathbf{0}_{m_1-1,m_3} \\ \mathbf{0}_{m_2-1,m_1} & \mathbf{Q}'_2 & \mathbf{0}_{m_2-1,m_3} \\ \mathbf{0}_{m_3-1,m_1} & \mathbf{0}_{m_3-1,m_2} & \mathbf{Q}'_3 \end{bmatrix}$$

is orthogonal so that $\mathbf{H}\mathbf{H}' = \mathbf{I}_m$ and that

$$\mathbf{H}\mathbf{R}\mathbf{H}' = \begin{bmatrix} \mathbf{B} & \mathbf{0}_{3,m-3} \\ \mathbf{0}_{m-3,3} & (1-\rho_0)\mathbf{I}_{m-3} \end{bmatrix},$$

where **B** is given by Equation (3A.5). Since **H** is orthogonal, we see that **R** is positive definite if and only if $\mathbf{H}\mathbf{R}\mathbf{H}'$ is positive definite. Because $\mathbf{H}\mathbf{R}\mathbf{H}'$ is symmetric, it is positive definite if and only if its leading principal minors are all positive; see, for example, p. 205 of Searle [1982]. Moreover, from p. 266 of Searle [1982], we see that

$$\det\begin{bmatrix} \mathbf{B} & \mathbf{0}_{3,m-3} \\ \mathbf{0}_{m-3,3} & (1-\rho_0)\mathbf{I}_{m-3} \end{bmatrix} = \det^{m-3}(\mathbf{B})\det^3[(1-\rho_0)\mathbf{I}_{m-3}] = (1-\rho_0)^{3(m-3)}\det^{m-3}(\mathbf{B})$$

(recall that $m \geq 6$); and thus **R** is positive definite if and only if $1-\rho_0 > 0$ and the leading principal minors of **B** are positive. Combining the inequality $1-\rho_0 > 0$ with the requirement that the first-order leading principal minor of **B** must be positive yields Equation (3A.1). Requiring the second-order leading principal minor of **B** to be positive

yields Equation (3A.2). Finally if we take ρ_0 and ρ_1 as given quantities satisfying (3A.1) and (3A.2), then Equation (3A.4) is a quadratic equation in ρ_2 having the form $a\rho_2^2 + b\rho_2 + c = 0,$ where the quadratic coefficient is given by $a = -m_1 m_3[1+(m_2-1)\rho_0],$ the linear coefficient is given by $b = 2m_1 m_2 m_3 \rho_1^2,$ and the intercept is given by

$$c = \left\{\prod_{i=1}^{3}[1+(m_i-1)\rho_0]\right\} - \rho_1^2 m_2 m_3[1+(m_1-1)\rho_0] - \rho_1^2 m_1 m_2[1+(m_3-1)\rho_0].$$

Because $a < 0,$ we must have $\det(\mathbf{B}) > 0$ for all $\rho_2 \in \left(\rho_2^*, \rho_2^{**}\right),$ where ρ_2^* and ρ_2^{**} are the roots of Equation (3A.4) and $\rho_2^* < \rho_2^{**}$. This completes the proof of Proposition 3A-1.

Remark 3A-1. In the proof of Proposition 3A-1, we are of course assuming that the quadratic equation $a\rho_2^2 + b\rho_2 + c = 0$ has discriminant $b^2 - 4ac > 0$. On the other hand if $b^2 - 4ac \le 0,$ then we cannot find any feasible real values of ρ_2 for which **R** will be positive definite. In the special case that all groups have the same size so that $m_1 = m_2 = m_3,$ we see that

$$b^2 - 4ac = 4m_1^2\left\{[1+(m_1-1)\rho_0]^2 - \rho_1^2 m_1^2\right\}^2 > 0$$

if Equation (3A.2) is satisfied; and thus when all groups have the same size, Proposition 3A-1 is guaranteed to provide a means of choosing feasible real values of ρ_0, ρ_1, and ρ_2 for which **R** will be positive definite. Moreover in the case of unequal group sizes, if we pick ρ_0 to satisfy Equation (3A.1) and if we take $\psi \in (-1,+1)$ such that $\rho_1 = \psi\rho_0$ satisfies Equation (3A.2), then it can be shown that the choice $\rho_2 = \psi\rho_1 = \psi^2\rho_0$ always ensures that **B** (and hence **R**) is positive definite. Although we are unable to prove that $b^2 - 4ac$ must be positive for all possible values of m_i with $m_i \ge 2$ for $i = 1, 2, 3$ and for all possible values of ρ_0 and ρ_1 that satisfy Equations (3A.1) and (3A.2), in our computational

experience we have always been able to find feasible real values of ρ_0, ρ_1, and ρ_2 for which **R** is positive definite using Equations (3A.1) through (3A.5) as given in Proposition 3A-1.

7. References

1. Armstrong and Associtates, 2009, Warehousing in North America – 2009.

2. Ballou, R.H., Rahardja, H., Sakai, N., 2002, Selected country circuity factors for road travel distance estimation, Transportation Research Part A, 36, 843–848.

3. Brimberg, J.,Hansen, P., Mladenovic, N.,Taillard, E.D. ,2000, Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem, Operations Research, 48, 444–460.

4. Bucci, M.J., Kay, M.G., Warsing, D.P., Joines, J.A., 2009, Metaheuristics for Facility Location with Economies of Scale, Fitts Department of Industrial and Systems Engineering working paper, North Carolina State University, Raleigh NC.

5. Chopra, S., Meindl, P., 2004, Supply Chain Management: Strategy, Planning, and Operation, Prentice Hall, NJ.

6. Cooper, L., 1963, Location-allocation problems, Operations Research, 11, 331–343.

7. Corbett, C.J., Rajaram, K., 2006, A Generalization of the Inventory Pooling Effect to Nonnormal Dependent Demand, Manufacturing & Service Operations Management, 8, 351−358.

8. Croxton, K.L., Zinn,W., 2005, Inventory Considerations in Network Design, Journal of Business Logistics, 26, 149–168.

9. Das, C., Tyagi, R., 1999, Effects of Correlated Demands on Safety Stock Centralization: Patterns of Correlation versus Degree of Centralization, Journal of Business Logisitics, 20, 205–213.

10. Dorey, M., Joubert, P., Vencatasawmy, C., 2005, Modeling Dependencies: An Overview, Finance & Investment Conference, Silverstone.

11. Daskin, M.S., Coullard, C.R., Shex, Z.M., 2002, An Inventory-Location Model: Formulation, Solution Algorithm and Computational Results, Annals of Operations

Research, 110, 83–106.

12. Daskin, M.S., 1995, Network and discrete location: models, algorithms, and applications, John Wiley and Sons, New York.

13. Eppen, G.D., 1979, Effects of Centralization of Expected Costs in a Multi-Location Newsboy Problem, Management Science, 25, 5, 498–501.

14. Erkip, N., Hausman, W.H., Nahmias, S., 1990, Optimized Centralized Ordering Policies in Multi-Echelon Inventory Systems with Correlated Demands, Management Science, 36, 3, 381–392.

15. Evers, P.T., Beier, F. J., 1993, The Portfolio Effect and Multiple Consolidation Points: A Critical Assessment of the Square Root Law, Journal of Business Logistics, 14, 2, 109–125.

16. Hansen, P., Mladenovic, N., 1997, Variable Neighborhood Search for the P-median, Location Science, 5, 207–226.

17. Kay, M.G., 2006, Matlog: Logistics Engineering Toolbox, North Carolina State University (http://www.ie.ncsu.edu/kay/matlog).

18. Lowe's Companies, Inc., 2006 Annual Report, http://www.shareholder.com/lowes/annual.cfm

19. Mahmoud, M.M., 1992, Optimal Inventory Consolidation Schemes: A Portfolio Effect Analysis, Journal of Business Logistics, 13, 193–214.

20. Maister, D.H., 1976, Centralization of Inventories and the 'Square Root Law,' International Journal of Physical Distribution, 6, 124–134.

21. Melachovsky, J., Prins, C., Calvo, R.W., 2005, A Metaheuristic to Solve a Location-Routing Problem with Non-Linear Costs, 11, 375–391.

22. Mirchandani, P.B., Francis, R.L., 1990, Discrete Location Theory, Wiley, New York.

23. Ozsen, L., Coullard, C.R., Daskin, M.S., 2008, Capacitated Warehouse Location Model with Risk Pooling, Naval Research Logistics, 55, 295–312.

24. Nozick, L.K., Turnquist, M.A., 1998, Integrating Inventory Impacts into a Fixed-Charge Model for Locating Distribution Centers, Transportation Research Part E, 34,

173–186.

25. Rardin, R.L., Uzsoy, R., 2001, Experimental Evaluation of Heuristic Optimizatoin Algorithms: A Tutorial, Journal of Heuristics, 7, 261–304.

26. Searle, S.R., 1982, Matrix Algebra useful for statistics, Wiley, New York.

27. Shen, Z.M., Coullard, C., Daskin, M.S., 2003, A Joint Location-Inventory Model, Transportation Science, 37, 40–55.

28. Snyder, L.V., Daskin, M.S., Teo, C.P., 2007, The Stochastic Location Model with Risk Pooling, European Journal of Operational Research, 179, 1221–1238.

29. Sourirajan, K., Ozsen, L., Uzsoy, R., 2008, A genetic algorithm for a single product network design model with lead time and safety stock considerations, European Journal of Operational Research, 2008, Article in Press.

30. Vidyarthi, N., Celebi, E., Elhedhli, S., Jewkes, E., 2007, Integrated Production-Inventory-Distribution System Design with Risk Pooling: Model Formulation and Heuristic Solution, Transportation Science, 41, 392–408.

31. Walgreen Company, 2006 Annual Report, http://www.shareholder.com/lowes/annual.cfm

32. Whitaker, R.A., 1985, Some Add-Drop and Drop-Add Interchange Heuristics for Non-Linear Warehouse Location, The Journal of the Operational Research Society, 36, 61–70.

33. Xu, K., Evers, P.T., 2003, Managing single echelon inventories through demand aggregation and the feasibility of a correlation matrix, Computers and Operations Research, 30, 297–308.

34. Zinn, W., Levy, M., Bowersox, D.J., 1989, Measuring the Effect of Inventory Centralization / Decentralization on Aggregate Safety Stock: The "Square Root Law" Revisited, Journal of Business Logistics, 10, 1-14.

CHAPTER 4: An Application of Heuristics Incorporating Economies of Scale to Facility Location Problems in Carpet Recycling

Michael J. Bucci[*], Ryan Woolard[†], Jeffrey Joines[†], Kristin Thoney[†], Russell E. King[*]

[*] **Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA**

[†] **Department of Textile Engineering/Chemistry/Science, North Carolina State University, Raleigh, NC 27695, USA**

Reprint of paper to be submitted to the Journal of the Textile Institute

An Application of Heuristics Incorporating Economies of Scale to Facility Location Problems in Carpet Recycling

Michael J. Bucci[*], Ryan Woolard[†], Jeffrey Joines[†], Kristin Thoney[†], Russell E. King[*]
[*] Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA
[†] Department of Textile Engineering/Chemistry/Science, North Carolina State University, Raleigh, NC 27695, USA

Abstract

The United States carpet industry has set a goal to divert 40% of used carpet from landfills by 2012. This significant volume, estimated to be 2.7 billion lb per year, requires the design of an effective logistic network to process the used materials. We investigate the use of facility location heuristics originally developed for the forward distribution of products to this reverse logistics system. The model includes transportation costs and fixed facility and processing costs at the processing plant, the latter exhibiting economies of scale as the facility size increases. The objective is to locate an unknown number of carpet recycling facilities to minimize the total cost. We evaluate the model using data from a known carpet collection network in the continental United States and compare these findings to models that assume a significant increase in collection locations and collection rates to meet specific carpet diversion targets. We show the impact of economies of scale on the solution structure and the impact that collection volumes have on the solution.

Keywords

Facility location, reverse logistics, carpet recycling, economies of scale

1. Introduction

The United States carpet industry has set a goal to divert 40% of used carpet from landfills by 2012 [CARE annual report 2007]. This significant volume, estimated to be 2.7 billion lb per year, requires the design of an effective logistic network to process the used materials. We investigate the use of facility location heuristics originally developed for the forward distribution of products to this reverse logistics system. The model attempts to minimize the total cost to locate an unknown number of carpet recycling facilities that collect carpet bales from a known collection network. The model includes transportation cost from the collection facility to the recycling facility and fixed facility costs and processing costs at the processing facility, the latter exhibiting economies of scale as the facility size increases. The model is designed to evaluate a known carpet collection network, the Carpet America

Recovery Effort (CARE) network, in the continental United States. The model also compares this network with a collection network that assumes a significant increase in collection locations and collection rates to meet specific carpet diversion targets. We show the impact of economies of scale on the solution structure and the impact that collection volumes have on the number of facilities in the network. Extensions of the model are also presented.

The remainder of the paper is organized as follows. In Section 2 we review the relevant literature; Section 3 describes the general model formulation; Section 4 explains the experimental design, solution approaches and computational results; and Section 5 offers conclusions and directions for future research.

2. Relevant Research

Reverse logistics systems have become more prominent as the desire/need to recycle materials has grown. Fleischmann et al. [2000] provide an excellent overview of product recovery systems, comparing them to one another and to traditional logistics structures. They characterize recent case studies, defining classifications for different types of recycling networks and comparing them with traditional forward logistics networks. Brito et al. [2003] review and characterize over sixty case studies on reverse logistics.

Specific to the carpet industry there are several papers relevant to this research. Louwers et al. [1999] applied a location-allocation model for carpet recycling in Europe. Their model considered both pre-processing centers, with capabilities for collection and sorting of carpet materials, and processors of the used carpet. Additionally, they consider a variety of costs, including storage costs at the pre-processing center, processing cost at the pre-processing center, transportation cost to the pre-processing center, and transportation cost to the processing center. Their objective is to find the number of pre-processing centers and their corresponding locations to minimize the total cost of operating the collection network. A

location-allocation heuristic is used as a solution approach. Realff et al. [1999] develop a mixed integer program model to maximize profit for a carpet recycling case study. Their model assesses profit based on revenue from selling the recycled carpet product minus the fixed facility costs, storage costs, collection and processing costs, and transportation costs. Their model includes economies of scale at the recycling center, presumed to be modeled as a piecewise linear function. They show the importance of processing center locations on the network economics as well as the importance of considering pre-processing (product separation) at the collection center to avoid shipping low value material that is not required to pass through the recycling plant process. Our model utilizes some of the cost modeling presented in this research. Biehl et al. [2007] simulate a reverse logistics network for carpet recycling, assessing the viability of the network based on significant variability in return flows and several other factors. They show the need for marked improvement in recycling rates and a reduction in variability of return flows for reverse logistics networks to meet the 40% target diversion rate.

A practical example of a carpet recycling network in the U.S. is the CARE network which is comprised of 58 collection facilities, as shown in Figure 1. For 2007 the CARE network diverted 296 million lb of carpet from landfills – of which 275 million lb was recycled. The CARE group expects a continued increase in collections as investments in reclamation and recycling increase. They cite Shaw Industries' 100 million-pound-capacity recycling facility for nylon-6 in Augusta, GA as an important investment in recycling technology [CARE 2007].

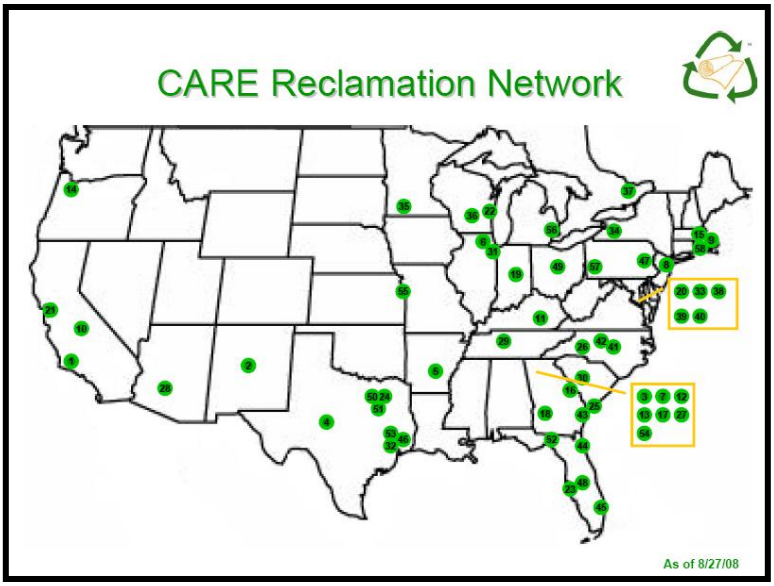

Figure 1: CARE Reclamation Network – [CARE 2009]

Facility location models for forward logistics have been widely studied over the last several decades [Mirchandani and Francis 1990, Daskin 1995]. In many situations this breadth of research can be readily applied to reverse logistics systems as the two networks have many common characteristics. Bucci et al. [2009] have developed meta-heuristics that can solve large uncapacitated location problems that exhibit economies of scale in inventory, production, and/or transportation. Their primary solution approach, shown in Figure 2 and adapted for this research, is a constructive ADD procedure combined with a discrete alternate location–allocation (ALA) procedure. The ADD procedure is used to sequentially add facilities, one at a time, to the solution. This method has been shown to be effective as the solution for *n* facilities tends to have many similarities with the solution with *n+1* facility solution [Daskin 1995]. The ALA procedure is a widely used facility location solution approach that was developed to separate the decision of assigning the location of facilities from the allocation of customers to those facilities [Cooper 1964]. The ALA procedure solves the two decisions separately, iterating between the two decisions until the solution no longer improves [Brimberg et. al 2000]. When the ALA procedure is combined

with the ADD procedure all of the facility locations can change within the location step, thus the *n*+1 solution may have an entirely unique set of facility locations from the *n*-facility solution.

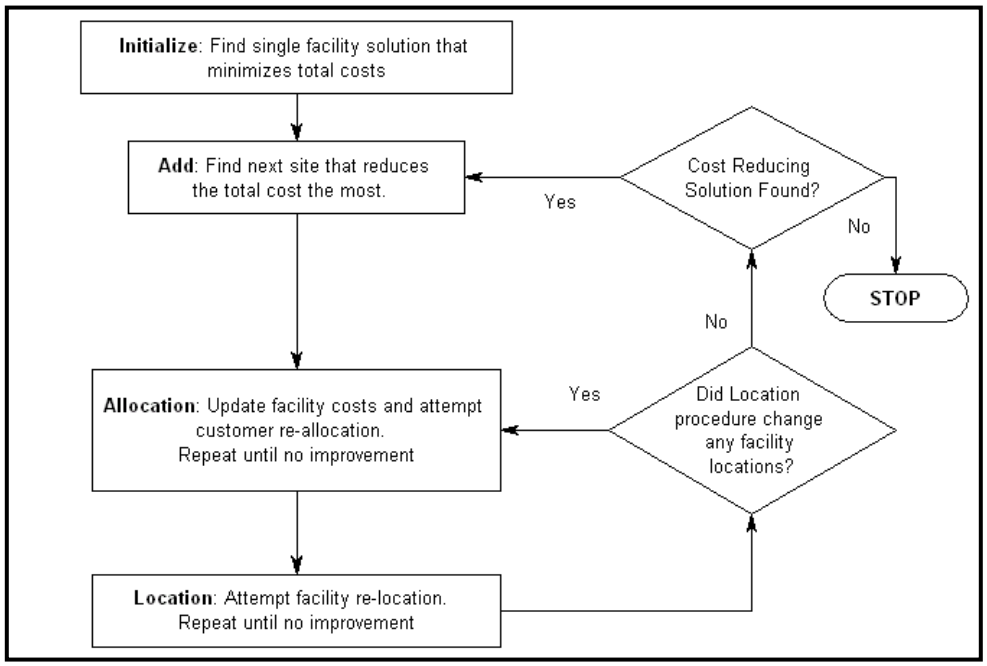

Figure 2: General Flow of the Solution Meta-heuristic [Bucci et al. 2009]

3. Model Formulation

The model is similar to the formulation of Bucci et al. [2009] and is defined as follows

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij} + \sum_{i=1}^{n} F_i Y_i + \sum_{i=1}^{n} C_i \quad S_i \quad S_i \tag{4.1}$$

subject to

$$\sum_{i=1}^{n} X_{ij} = 1 \text{ for } j = 1, 2, n \tag{4.2}$$

$$S_i = \sum_{j=1}^{n} X_{ij} D_j \tag{4.3}$$

$$Y_i - X_{ij} \geq 0 \text{ for } i = 1, 2 \ldots, m \;\; j = 1, 2 \ldots, n \tag{4.4}$$

$$X_{ij} \in \; 0,1 \;\; for \; i = 1, 2, \ldots, m \;\; j = 1, 2, \ldots, n \tag{4.5}$$

$$Y_i \in \; 0,1 \;\; \text{for } i = 1, 2, \ldots, m \tag{4.6}$$

where

$X_{ij} =$ a binary variable set to 1 if recylcing facility i collects product from collection center j

$Y_i =$ a binary variable set to 1 if recycling facility i is open, and 0 otherwise

$C_{ij} =$ transportation cost from collection center j to recycling facility i

$D_j =$ the demand at collection center j

$F_i =$ the fixed operating cost of recycling center i

$S_i =$ size of recycling facility i

$C_i(S_i) =$ marginal unit cost for recycling facility i as a function of its size, S_i

The first term in the objective function (4.1) computes the total transportation costs from the collection centers to the recycling facilities. The second term computes the fixed costs for locating the recycling facilities while the third term computes the system-wide processing costs, the latter being a function of the facility size. Constraints (4.2) require that each collection center ships product to only one recycling facility while constraints (4.3) ensure that the capacity of the recycling center equals the collection center demand allocated to it. Constraints (4.4) state that a recycling center must be open ($Y_i = 1$) if it receives product from at least one collection center (X_{ij}=1 for some j=1,2,…,n). Constraints (4.5) and (4.6) are integer constraints.

3. Case Study

The model in the case study includes fixed facility costs, transportation costs, and processing costs at the facility, the latter exhibiting economies of scale as the facility size increases. For the first part of the case study, the CARE collection network was used to represent the collection center network. The centroid of the five-digit ZIP code in which the CARE

facility is located was assumed to be its precise location for modeling purposes. Because the percentage of carpet collected at each CARE location was not available to us, we assumed the 2007 annual total of 296 million lb of collected carpet was allocated among the different CARE locations based on the relative populations near each location. Using U.S. census population data for five-digit ZIP codes in the continental United States (31,569 in total), we summed the populations of all ZIP codes within a 24-mile radius of the CARE network location and used this as a demand weight for each location. The 24-mile radius was obtained as a probable radius from which the collection center would draw demand based on pickup/delivery costs versus landfill tipping fees (Repa 2005). Other radii, 12 miles and 48 miles, were investigated and were shown to have little impact on the model results.

The possible recycling facilities locations were chosen to be the 877 three-digit ZIP codes in the continental United States. Transportation cost is annualized as one-way truckload shipments of unprocessed bales of carpet from the collection center to the recycling facility. The transportation cost was calculated by multiplying a $2.32/mile charge [Realff et al. 1999] by the estimated road distance between the customer and facility locations and the number of truckloads shipped per year [Bucci et al. 2009]. The number of truckloads was calculated by a conversion of carpet weight into truckloads (59,400 lb / truckload) [Realff 2006]. The fixed annual facility cost was estimated from to be $1M [Realff et al. 1999]. The processing cost per pound of carpet exhibits economies of scale such that the processing cost per pound decreases at certain breakpoints as the facility size increases [Realff et al. 1999]. We assume the change in processing cost is linear between the breakpoints, but we allow for the processing cost to plateau near the breakpoints. The assumption here is that the operation manager will build the facility to a slightly larger size and have idle capacity, if the economics justify this as opposed to building a more expensive facility to the precise size needed. For comparison, we also investigate models without economies of scale, using a linear cost for recycling each pound of carpet (Figure 3). The constructive ADD procedure combined with a discrete alternate location–allocation (ALA) procedure described by Bucci et al. [2009] is used to solve all of the models.

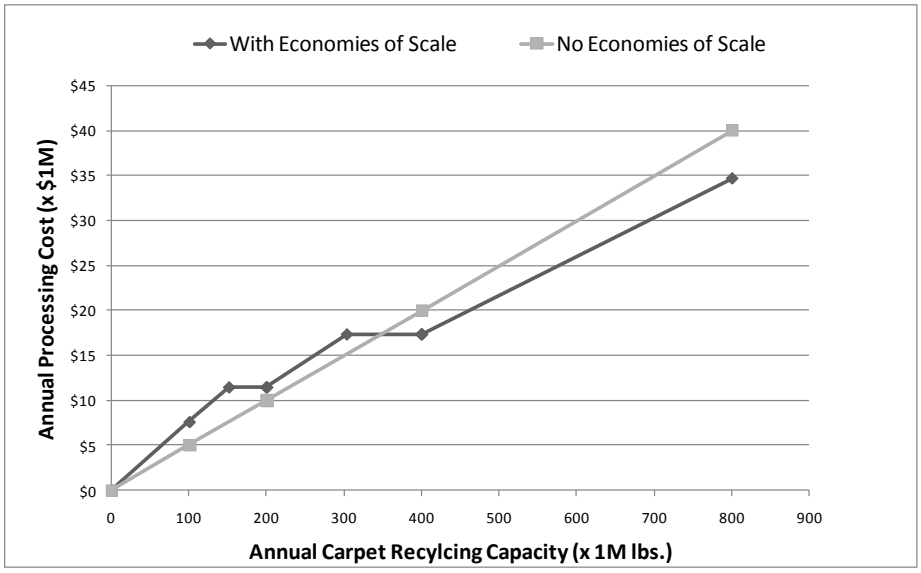

Figure 3: Recycling Center Processing Costs as a Function of Facility Size

Since the CARE network is currently capturing much less than its 40% diversion target, for the second part of the case we create a hypothetical collection network that assumes a significant increase in collection locations and collection rates to meet the carpet diversion target. For this model we assume the collection center locations are the 400 largest three-digit ZIP codes, by population, in the continental United States. The location of each of the three-digit ZIP codes is taken to be the population weighted centroid of its constituent five-digit ZIP codes. The demand weight for each of these locations is calculated based on the population of the three-digit ZIP code in which the collection center is located. For the larger network several collection volumes are investigated: 300 M, 600 M, 1200 M, 1800 M, and 2400 M lb. Our interest here is the impact that the fixed facility costs and the processing economies of scale at the recycling facility will have on the recommended location and size of the recycling facilities. Figure 4 shows the 400 collection center locations for this model. We also assume the 400 collection center locations are also the possible recycling facility locations.

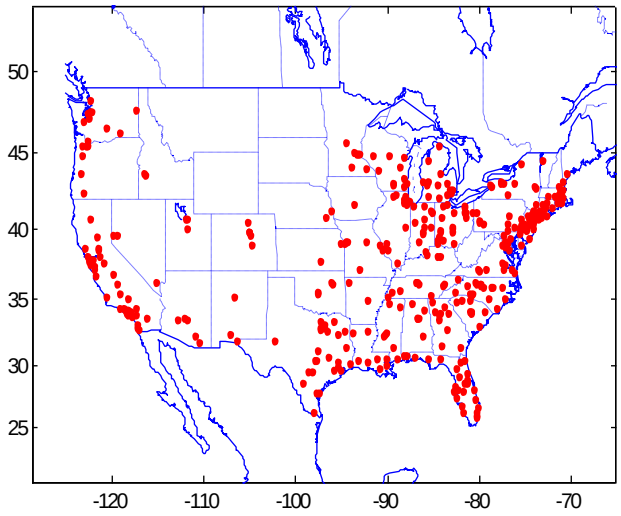

Figure 4: Collection Center Locations for 400 Customer model

3.1 Results

The model in the case study includes fixed facility costs, transportation costs, and processing costs at the facility, the latter exhibiting economies of scale as the facility size increases. For the first part of the case study, the CARE collection network was used to represent the collection center network. The centroid of the five-digit ZIP code in which the CARE facility is located was assumed to be its precise location for modeling purposes. Because the percentage of carpet collected at each CARE location was not available to us, we assumed the 2007 annual total of 296 million lb of collected carpet was allocated among the different CARE locations based on the relative populations near each location. Using U.S. census population data for five-digit ZIP codes in the continental United States (31,569 in total), we summed the populations of all ZIP codes within a 24-mile radius of the CARE network location and used this as a demand weight for each location. The 24-mile radius was obtained as a probable radius from which the collection center would draw demand based on

pickup/delivery costs versus landfill tipping fees (Repa 2005). Other radii, 12 miles and 48 miles, were investigated and were shown to have little impact on the model results.

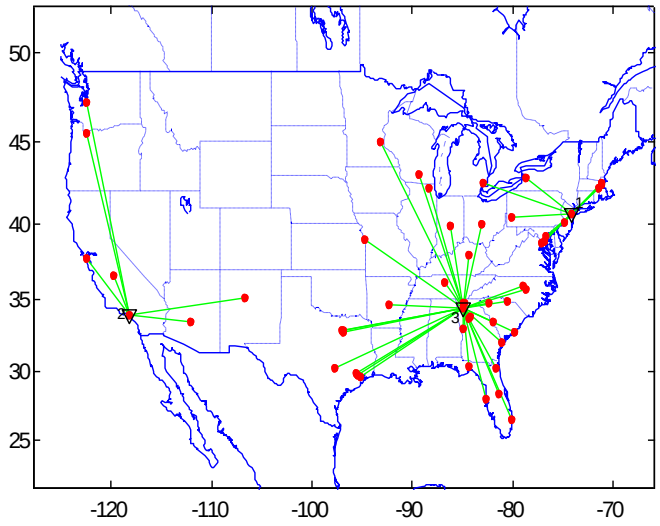

Figure 5: CARE network solution with no economies of scale in processing costs

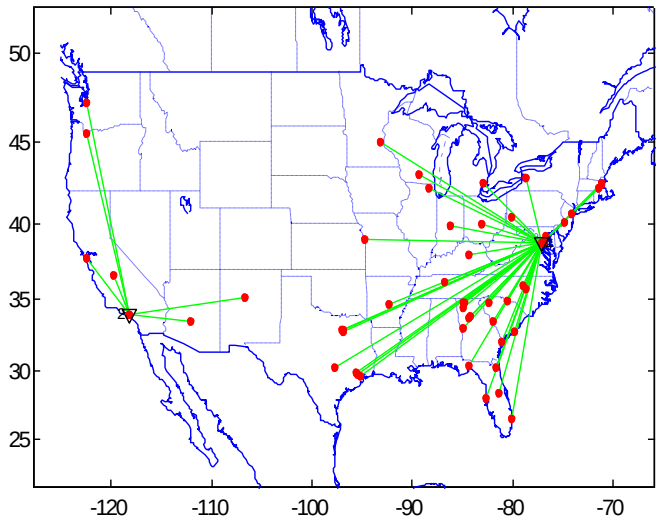

Figure 6: CARE network solution with economies of scale in processing costs

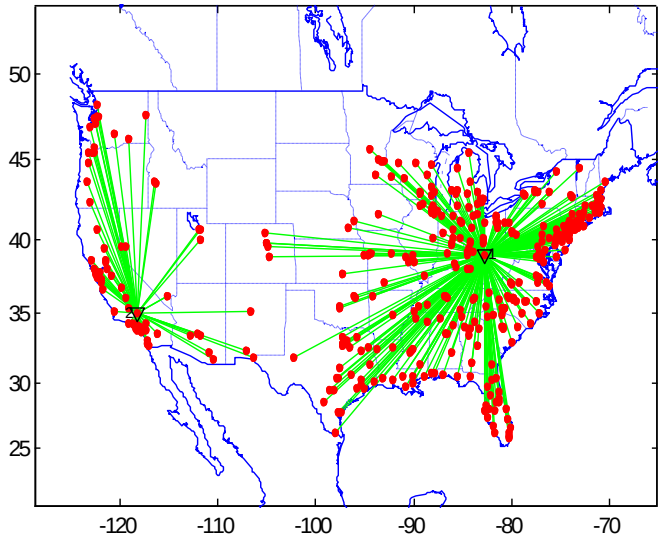

Figure 7: 400-customer solution with 300M lb of demand, and economies of scale in processing costs

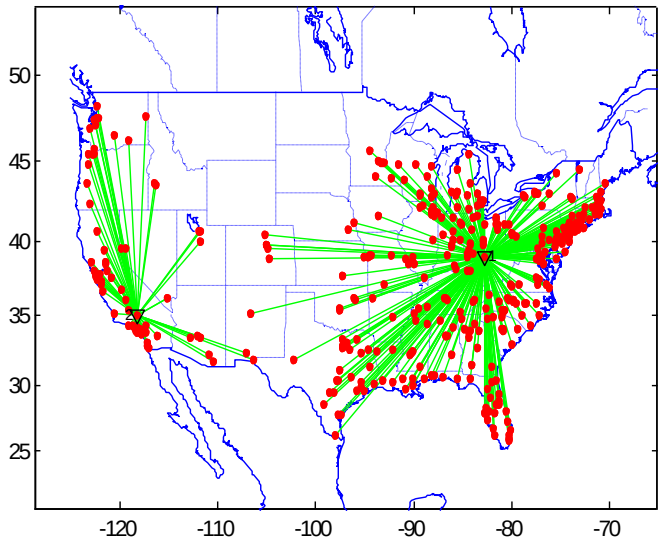

Figure 8: 400-customer solution with 600M lb of demand, and economies of scale in processing costs

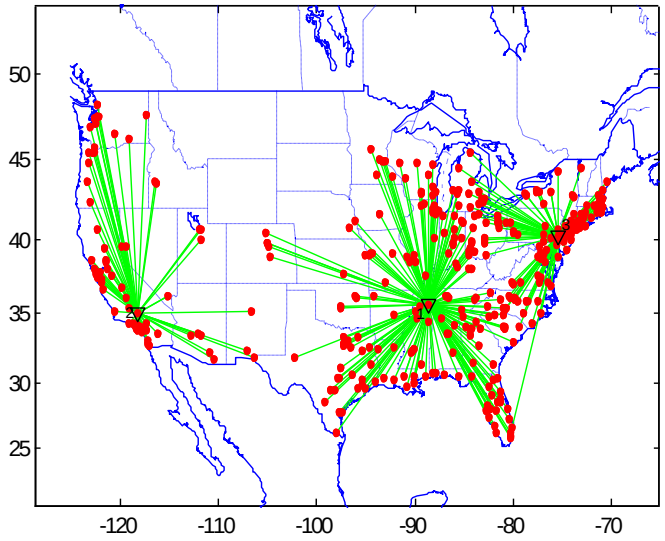

Figure 9: 400-customer solution with 1200M lb of demand, and economies of scale in processing costs

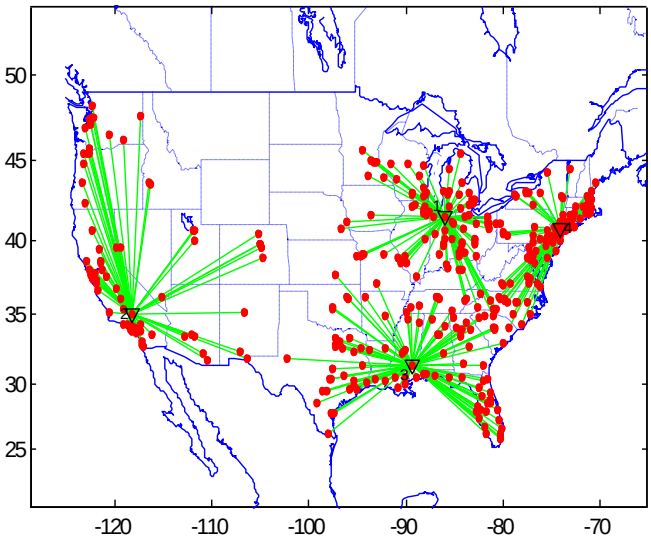

Figure 10: 400-customer solution with 1800M lb of demand, and economies of scale in processing costs

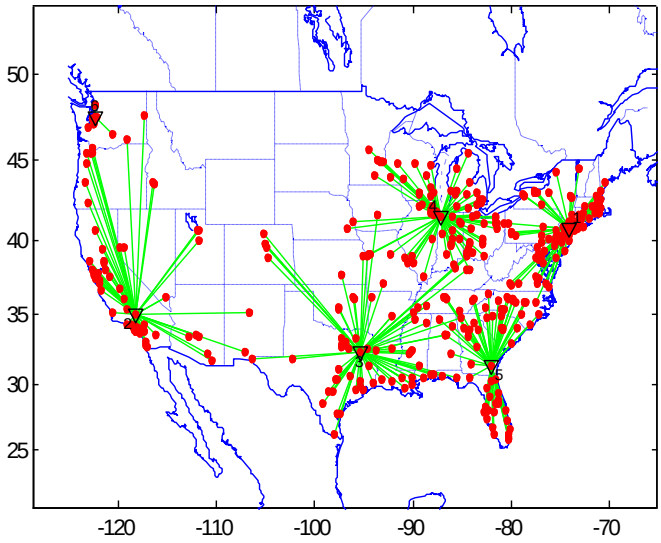

Figure 11: 400-customer solution with 2400M lb of demand, and economies of scale in processing costs

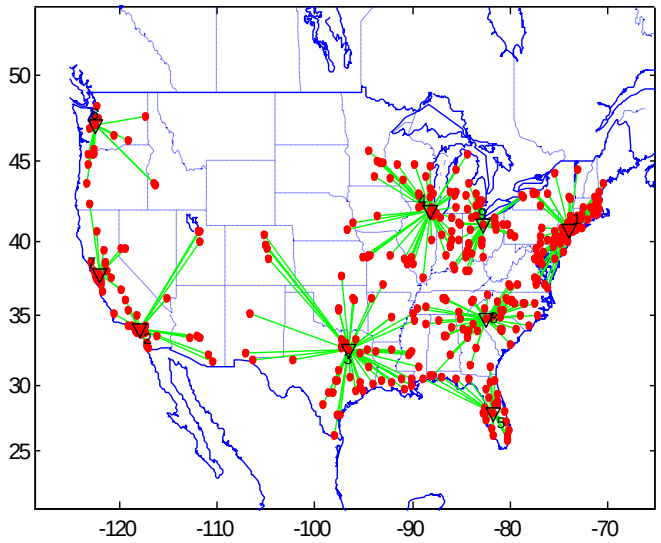

Figure 12: 400-customer solution with 1800M lb of demand, and no economies of scale in processing costs

Table 1: Summary of Recommended Solution for each Model Studied

Model	# of Facilities	Facility Capacities (M lb)
With Economies of Scale:		
CARE Network, 296 M lb	2	227, 69
400 customers, 300 M lb	2	233, 67
400 customers, 600 M lb	2	467, 133
400 customers, 1200 M lb	3	530, 269, 402
400 customers, 1800 M lb	4	428, 427, 475, 470
400 customers, 2400 M lb	6	577, 483 401, 492, 400, 47
No economies of Scale:		
CARE Network, 296 M lb	3	111, 69, 116
400 customers, 1800 M lb	9	395, 220, 256, 235, 137, 71, 101, 205, 181

4. Conclusions and Directions for Future Work

The United States carpet industry has set a goal to divert 2.7 billion lb of carpet per year from landfills, requiring an effective logistic network to process the used materials. In this paper we investigate the use of facility location heuristics originally developed for the forward distribution of products to this reverse logistics system. The model attempts to minimize the transportation cost, fixed facility costs and processing costs at the processing facility, the latter exhibiting economies of scale as the facility size increases. We evaluated a known carpet collection network as well as a hypothetical network that assumes a significant increase in collection locations and collection rates to meet the carpet diversion targets. We show the impacts economies of scale have on the solution structure and the impact collection volumes have on the number of facilities in the network.

There are several avenues for future research. Investigating sorting options and their associated costs at the collection center would offer more insights into the type of network that should be designed. Capacitating the recycling centers may also yield important changes in the solutions. Integrating the forward logistic network with the reverse network will improve the understanding of the total cost of a closed loop system.

5. Acknowledgments

This research is supported, in part, by the Institute for Textile Technology

6. References

1. Biehl, M., Prater, E., Realff, M.J., 2007, Assessing performance and uncertainty in developing carpet reverse logistics systems, Computers and Operations Research, 34, 443-463.

2. Brimberg, J.,Hansen, P., Mladenovic, N.,Taillard, E.D. ,2000, Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem, Operations Research, 48, 444–460.

3. Bucci, M.J., Kay, M.G., Warsing, D.P., Joines, J.A., 2009, Metaheuristics for Facility Location with Economies of Scale, Fitts Department of Industrial and Systems Engineering working paper, North Carolina State University, Raleigh NC.

4. Cooper, L., 1963, Location-allocation problems, Operations Research, 11, 331–343.

5. CARE (Carpet America Recovery Effort) annual report, 2007, http://www.carpetrecovery.org/pdf/annual_report/07_CARE-annual-rpt.pdf

6. CARE (Carpet America Recovery Effort) network website, 2009, http://www.carpetrecovery.org/pdf/reclamation_centers/Carpet_Reclamation_Centers.pdf

7. Daskin, M.S.,1995, Network and discrete location: models, algorithms, and applications, John Wiley and Sons, New York.

8. de Brito, M.P., Dekker, R., Flapper, S.D.P., 2003, Reverse Logistics – a review of case studies, ERIM Report Series.

9. Fleishchmann, M., Krikke, H.R., Dekker, R., Flapper, S.D.P., 2000, A characterization of logistics networks for product recovery, Omega, 28, 653-666.

10. Louwers, D., Kip, B.J., Peters, E., Souren, F., Flapper, S.D.P., 1999, A facility location allocation model for reusing carpet, Computers and Industrial Engineering, 36, 855-869.

11. Mirchandani, P.B., Francis, R.L., 1990, Discrete Location Theory, Wiley, New York.

12. Realff, M.J., Ammons, J.C., Newton, D., 1999, Carpet Recycling: Determining the Reverse Production System Design, Polymer-Plastics Technology and Engineering, 38:3, 547-567.

13. Realff, M., Systems Planning for Carpet Recycling. 2006, Recycling in Textiles, Editor Youjiang Wang, CRC Press.

14. Repa, E.W., 2005, NSWMA's 2005 Tip Fee Survey, National Solid Wastes Management Association Website, http://wastec.isproductions.net/webmodules/webarticles/articlefiles/478-Tipping%20Fee%20Bulletin%202005.pdf

CHAPTER 5: Conclusions and Directions for Future Work

5.1 Conclusions

This dissertation has focused on proposing and formulating models of logistic network design problems, and demonstrating effective procedures for solving them. In particular, the models and methods presented in this research reflect a more dynamic supply chain environment, one in which traditionally segregated decision areas—strategic, tactical, and operational—are integrated and re-visited regularly. This research begins to address such integrated solution environments through the incremental development of heuristic solution procedures that can solve large-scale facility location problems, particularly those that include economies of scale in the unit cost of processing the items flowing through the network, and/or economies of scale in the costs of holding safety stock of these items to buffer against uncertainty in demand. Combining and extending existing heuristic approaches, several meta-heuristics were developed to solve location problems that exhibit those scale economies, modeled using a non-linear concave cost function. Using solution time and newly developed measures of computational effort, we showed the resulting solution methods offer near optimal solutions with moderate computational effort.

These meta-heuristics were then applied to a focused study on the use of different approximations to represent safety stock inventory costs in location models. In models with no inter-customer correlation of demand, the "Square Root Law," SRL_1,was shown be inaccurate at estimating inventory costs, primarily due to the equal allocation assumption upon which it is based. A more general concave cost function, SRL_2, which accounts for facility size more explicitly, was shown to consistently outperform SRL_1 with minimal additional computational effort. To assess scenarios with inter-customer demand correlation, a straightforward methodology was developed to produce a valid correlation matrix under the premise that correlation between pairs of customers decreases as the distance between the customer pair increases. In our study of models with low and

moderate inter-customer correlation of demand, both SRL_1 and SRL_2 provide poor estimates of the inventory required and have significantly different underlying solution structures relative to SRL_0.

In the final chapter the aforementioned meta-heuristic solution procedures are then applied to an industrial problem, namely to locate facilities that recycle used carpet collected through a network of collection points dispersed throughout the continental U.S. The recycling facilities in this network exhibit economies of scale in processing costs. This industry application analyzes an existing, smaller-scale used carpet collection network and a larger, hypothetical collection network with demand and collection points spread more extensively across the continental U.S. The analysis provides insight into the number of recycling facilities that should be located and their respective size to support the current collection network and to support more extensive networks that represent future collection and reprocessing activity. The results also compare solutions for models whose processing costs reflect economies of scale and those that do not, highlighting the impact of their inclusion on the nature of the solutions.

5.2. Future Work

This research proposes and demonstrates the viability of formulating and solving integrated models of strategic, tactical, and operational decision-making in supply chain network design. There are many avenues to pursue related to this research. Some broad research areas that represent clear research opportunities are as follows:

- Combining heuristic approaches with optimization techniques and/or simulation (i.e., computationally intensive objective function evaluations—e.g., solving a VRP for each reallocation of demand or running a simulation that incorporates much more complex scale-economy costs than can be modeled via a simple concave cost function).

- Development of models capable of assessing frequent reconfiguration of the supply chain—especially as driven by updated information about customer demand and its effect on relative demand levels, CV values, and/or correlation values, thereby affecting the level of safety stock pooling benefits and possibly impacting the best allocation for a given set of locations and/or the actual location structure itself.
- Development of models and tools that can process large amounts of supply chain data of the type that is now commonly available in supply chain databases / software. (This issue is related to the bullet above, but more broadly considers the processing of data from the supply chain information system and the integration of this data into the solution models in a seamless fashion.)

APPENDICES

Appendix A: Supplemental Information for Chapter 2

A.1.0 Reprint of Journal Paper: IERC 2006

Title: A Modified ALA Procedure for Logistic Network Designs with Scale Economies

Authors: Michael J. Bucci and Michael G. Kay
Fitts Department of Industrial and Systems Engineering
North Carolina State University
Raleigh, NC 27695-7906

Reprint of paper published in proceedings for: Industrial Engineering Research Conference, IERC 2006, Orlando, Florida

A Modified ALA Procedure for Logistic Network Designs with Scale Economies

Michael J. Bucci and Michael G. Kay
Fitts Department of Industrial and Systems Engineering
North Carolina State University
Raleigh, NC 27695-7906

Abstract

Logistics network design problems involving locating factories and warehouses, assigning demand and supply to facilities, and selecting the mode and frequency of transport are typically modeled as large-scale integer programs. Product, inventory, and transportation costs are considered, but these models become difficult to solve when these costs are nonlinear. An alternative approach is to modify the alternate location-allocation (ALA) heuristic procedure developed by Cooper. This paper describes a modified ALA model for solving a logistic network design with nonlinear production scale economies. A range of economy of scale factors and production-to-transport cost ratios are considered for the model.

Keywords: ALA, location-allocation, scale economies, heuristics

1. Introduction

The logistics network design problem can involve determining the locations of factories and warehouses in the network, the assignment of suppliers to the factories and customers to warehouses, the selection of the mode and frequency of transport between all of the nodes in the network, and the amount of safety stock to be held at each node in the network. These problems are typically modeled as large-scale mixed-integer linear programs [1, pp. 82] or as nonlinear integer programs [2]. Product, inventory, and transportation costs are considered, but these models are not tractable when nonlinear economies of scale for either transportation, production, or safety stock (aggregation savings) are considered. Piecewise linear approaches can approximate these concave costs, but add significant complexity to the models [1, pp. 22].

An alternative approach is to utilize the alternate location-allocation (ALA) heuristic procedure developed by Cooper [3], where, for a given number of nodes, (continuous)

locations are determined followed by an assignment procedure, repeating until a locally optimal solution is found. The ALA procedure starts by selecting random initial facility locations. The allocation procedure assigns customer demand to the facility that is closest to the customer locations. In more complex logistic network designs a minimum cost network flow solution can be used to determine the allocation (e.g., Cooper [4], where the "location-transportation problem" is considered). After all demand is allocated, the location procedure is executed by finding the optimal location for each facility using a multi-dimensional unconstrained nonlinear search method. The allocation and location cycles are repeated until no further improvement is made, corresponding to a local optimum. The only cost consideration in this model is transportation cost. The ALA procedure can then be repeated with a new set of random start locations to increase the likelihood of finding the global optimum facility locations and demand allocations. Employing random starting locations for the initial assignment the ALA procedure been shown to be efficient at reaching good solutions after a small number of a runs for a facility location and allocation logistic network model with a limited number (less then 25) of facilities [5].

One extension of the ALA procedure is to account for scale economies in production. Rumelt [6] shows that scale economies in manufacturing are typically represented with an exponential function:

$$C = aS^b \tag{1}$$

where C is the cost of one unit of production, a is a constant for the unit cost at a scale of 1, S is the size of the facility, and b is the scale exponent. When $b < 0$, increases in facility size reduce the unit cost. This relationship can also be expressed as a ratio:

$$C_1 / C_2 = (S_1 / S_2)^b \tag{2}$$

where C_1 and C_2 are the unit costs in facilities of size S_1 and S_2, respectively. Rumelt estimates the scale economy exponent in manufacturing facilities is –0.35, and for the U.S. paper industry the exponent is between –0.47 and –0.56 [6]. A critical factor when

considering scale economies is the ratio of production cost per unit and transportation cost per unit (the "p/t ratio") for the logistic network. A large difference in these two costs will cause the model to be dominated by either the transportation or the production cost.

This paper explores the adaptation of the ALA procedure to solve a logistic network design problem that includes nonlinear production scale economies. A range of scale exponents and p/t ratios are investigated. As there has been little research using the ALA technique to solve this type of logistic network design problem, the model developed is extensively discussed along with analysis, conclusions, and directions for future work.

2. Implementation and Experimental Results

2.1 Model Parameters

The model was created and executed in MATLAB utilizing Matlog [7] as a building block for the models. The great circle distance in statute miles was used to determine the distance between facilities and customer locations. The location procedure is executed using the best of a Nelder-Mead direct search and a gradient based search. The first location-allocation cycle follows the standard ALA procedure. The location procedure remains the same for the additional cycles; however, the allocation cycle includes economies of scale, as a production cost per unit, based on the size of each facility from the previous allocation cycle (Eq. 2). The production cost is added to the transportation cost and the customer demand is then allocated to the facility with the lowest total cost. The procedure repeats until no improvement is made in lowering total cost. This modified ALA procedure (eosALA) is executed for 40 runs with random starting locations to assess the repeatability of the procedure.

2.2 Implementation

A single-source single tier supply chain was created with retail customer locations and production factories. Table 1 provides details on the supply chain and model parameters.

The p/t ratios are created using production unit cost in a facility that has 100% of retail demand (significant production scale advantages) and a transportation distance equal to the maximum distance between retail locations (in this dataset this is 2825 miles). A p/t ratio of one means a production cost of one unit equals the transportation cost for one unit to travel 2825 miles. The eosALA model was executed at each scale exponent and p/t ratio value for 40 repetitions. A second set of 40 repetitions (Run 2) at each setting was executed with a different random number seed to further assess the repeatability of the procedure.

Table 1: Supply Chain Parameters

	Parameter	Value	Comment
1	Retail locations	877	Center of population of 3-digit zip codes for continental U.S.
2	Retail demand	range	Based on population of 3-digit area
3	Number of factories	6	Set to give meaningful results for the scale exponents and p/t ratios chosen
4	Circuity factor	1.2	Used to convert great circle distances to estimated road distances
5	Scale exponents	range	Range used: 0,–0.1,–0.2,–0.3,–04,–0.5,–0.6
6	p/t ratios	range	Range used: 0, 0.2, 0.6, 0.8, 1, 2, 4, 6, 8
7	Product unit cost	$1	This is the unit production cost in the maximum facility size (100% of demand)

An example of the progression of the location-allocation ALA procedure is shown in Figures 1–3 for a scale exponent of –0.4 and a p/t ratio of 0.16. The red dots are the customer locations, black dots are the production facility locations, and green lines depict which facility serves each customer. Figure 1, with random start locations, depicts all six facilities having some retail allocation. The total cost for this cycle is $\$3.59 \times 10^6$.

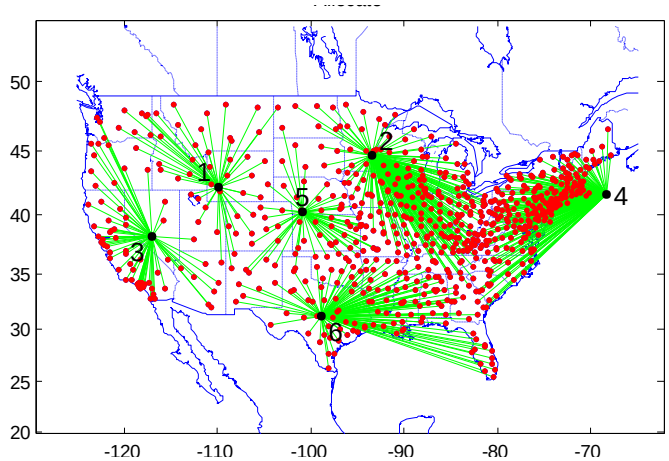

Figure 1: Example of an initial facility location-allocation

Figure 2 shows the progression of the procedure after four location-allocation cycles. Facilities 1 and 5 have no allocation as a result of the "greedy behavior" of the scale exponent in the allocation procedure in which larger facilities (lower cost per unit) increase their allocation at the expense of smaller facilities. Additionally, the four facilities with some allocation have migrated to more cost efficient locations as a result of the location procedure. The total cost for this cycle is $\$3.12 \times 10^6$.

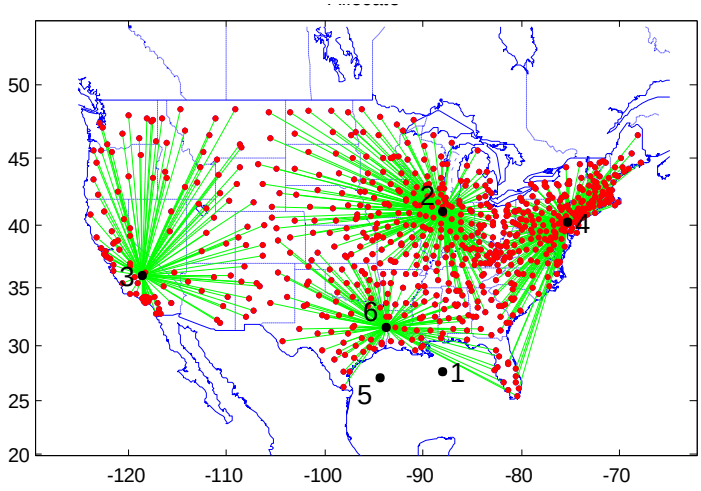

Figure 2: Example of intermediate facility location-allocation

Figure 3 shows the final solution, after 17 additional location-allocation cycles, with five of the facilities having 100% of the allocation and facility 1 having no allocation. The random locating of facilities with no allocation has resulted in facility 5, which had no allocation in Figure 2, finding a location that take demand from the larger facilities, in this case facility 2, 3, and 6. The additional location cycles resulted in minor improvements in facility locations from Figure 2 as the majority of the allocation has not changed between these cycles. The total cost for this cycle is $\$3.11 \times 10^6$.

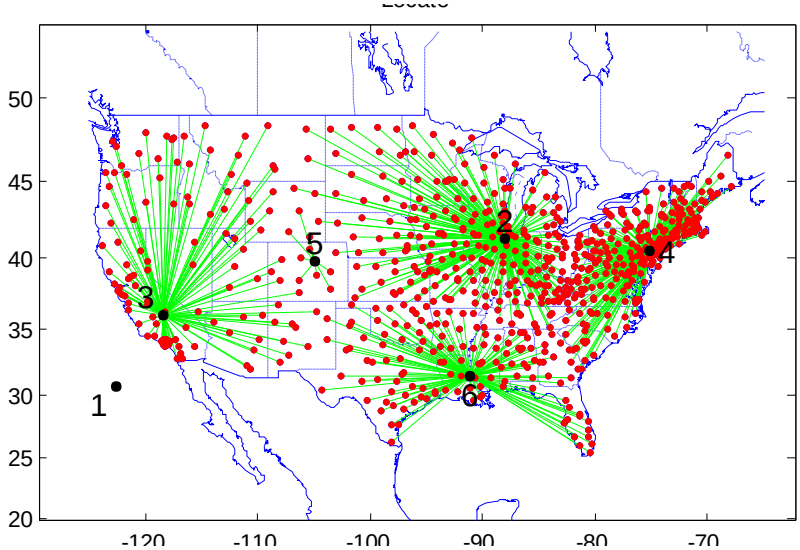

Figure 3: Example of final facility location-allocation

2.3 Results

Total cost for each of the 40 repetitions was used for comparing the solutions. The repeatability of the procedure with different random number seeds can be seen by comparing the results of Run 1 and Run 2 for the two measures used to assess the results. First, for the lowest cost run of the 40 repetitions the largest facility allocation percentage was analyzed to determine the impact of the scale exponent and p/t ratios on the allocation of demand. These results are shown in Tables 2 and 4 for Run 1 and Run 2 respectively. Second, the percentage of repetitions within 5% of the best run was used to gauge the repeatability of the

eosALA procedure in finding a solution close to the lowest cost solution. These results are shown in Tables 3 and 5 for Run 1 and Run 2, respectively. A precision level of 5% value was deemed reasonable for this type of problem.

Table 2: Run 1, largest facility percentage for lowest cost solution

		production cost vs. transportation cost (p/t) ratio							
		0	**0.02**	**0.04**	**0.08**	**0.16**	**0.3**	**0.6**	**1.2**
	0	0.243	0.246	0.246	0.246	0.246	0.246	0.246	0.246
	–0.1	0.244	0.246	0.245	0.247	0.264	0.272	0.303	0.780
	–0.2	0.246	0.246	0.246	0.245	0.246	0.443	0.803	0.825
scale	**–0.3**	0.246	0.246	0.246	0.251	0.256	0.519	0.776	1.000
exponent	**–0.4**	0.245	0.246	0.271	0.272	0.560	0.803	0.873	1.000
	–0.5	0.244	0.255	0.255	0.441	0.633	0.797	0.791	1.000
	–0.6	0.245	0.246	0.258	0.623	0.801	0.881	1.000	1.000

Table 3: Run 1, percentage of repetitions within 5% of the best run

		production cost vs. transportation cost (p/t) ratio							
		0	**0.02**	**0.04**	**0.08**	**0.16**	**0.3**	**0.6**	**1.2**
	0	97.5	100.0	100.0	95.0	100.0	100.0	100.0	100.0
	–0.1	97.5	100.0	100.0	100.0	90.0	97.5	100.0	100.0
	–0.2	100.0	97.5	97.5	92.5	100.0	100.0	100.0	100.0
scale	**–0.3**	100.0	97.5	100.0	100.0	100.0	97.5	100.0	92.5
exponent	**–0.4**	100.0	85.0	100.0	97.5	100.0	95.0	92.5	95.0
	–0.5	100.0	100.0	100.0	100.0	97.5	72.5	67.5	77.5
	–0.6	100.0	90.0	85.0	100.0	100.0	97.5	55.0	37.5

Table 4: Run 2, largest facility weight for lowest cost solution

		production cost vs. transportation cost (p/t) ratio							
		0	**0.02**	**0.04**	**0.08**	**0.16**	**0.3**	**0.6**	**1.2**
	0	0.245	0.246	0.245	0.246	0.246	0.246	0.246	0.246
	–0.1	0.245	0.246	0.246	0.246	0.264	0.245	0.365	0.673
	–0.2	0.244	0.246	0.243	0.239	0.262	0.266	0.621	0.804
scale	**–0.3**	0.245	0.246	0.246	0.271	0.258	0.639	0.809	1.000
exponent	**–0.4**	0.243	0.266	0.246	0.304	0.435	0.791	0.803	1.000
	–0.5	0.245	0.246	0.272	0.278	0.780	0.807	1.000	1.000
	–0.6	0.246	0.242	0.253	0.456	0.633	0.781	1.000	1.000

Table 5: Run 2, percentage of repetitions within 5% of the best run

		production cost vs. transportation cost (p/t) ratio							
		0	**0.02**	**0.04**	**0.08**	**0.16**	**0.3**	**0.6**	**1.2**
	0	97.5	97.5	100.0	100.0	100.0	97.5	100.0	100.0
	–0.1	95.0	100.0	95.0	90.0	92.5	100.0	100.0	100.0
	–0.2	97.5	100.0	100.0	95.0	97.5	100.0	97.5	100.0
scale	**–0.3**	97.5	97.5	100.0	97.5	97.5	97.5	80.0	35.0
exponent	**–0.4**	97.5	92.5	100.0	100.0	100.0	100.0	92.5	35.0
	–0.5	100.0	97.5	72.5	100.0	100.0	100.0	75.0	85.0
	–0.6	100.0	95.0	97.5	100.0	97.5	85.0	95.0	72.5

2.4 Analysis

In Tables 2 and 4 the standard ALA procedure (no production costs) corresponds to the values shown in row 1 and column 1. These results show that no facility has more then 25% of the allocation. There is little change in this maximum facility percentage for a range of scale exponent and p/t ratio values in the upper and left sides of the tables. Additionally, at the lower right corner of these tables, the scale exponent and p/t ratios cause the model to

gravitate to a single facility (i.e., 100%) solution. These areas of the table are interesting but do not offer very useful findings. However, the parameter settings in the center of the table show a significant impact on the maximum facility weight while maintaining multiple facility solutions. As in Tables 2 and 4, in Tables 3 and 5 all of the values in row 1 and column 1 represent the standard ALA procedure. On average, this procedure is within 5% of the best value in 98.9% of the repetitions. Looking at the center of the tables, for scale exponent values between –0.1 to –0.6 and p/t ratios between 0.08 and 0.03, the procedure is within 5% of the best value in 96.9% of the repetitions. The repeatability of the eosALA function for the center of Tables 3 and 5 shows promise that a modified ALA procedure can be used to model scale economies accurately for a range of scale exponents and p/t ratios. The lower repeatability values in the bottom right corner of Table 3 and 5 are primarily due to the eosALA procedure aborting once a single facility has over 99% of the demand.

2.5 Future Work

As there has been little research to modify the standard ALA procedure even using a simple single tier logistic network, there is a wide range of future work in this area. The current eosALA model needs verification for its efficiency with a variety of logistic network examples and should be compared with an equivalent solution methodology such as a mixed integer linear program with piecewise linear approximations for the nonlinear production scale economies along with a set of discrete facility locations. The model should be studied to determine the impact of scale economies on the number of allocation-location cycles required to find a solution and a more precise measure of the repeatability of the algorithm. The model can be extended to include additional variables such as fixed facility costs as well as additional nonlinear costs like transportation and safety stock (aggregation savings) scale economies. Minimum facility size and capacitated facility situations can be modeled as well as a combination of existing and new facilities. A distribution system design can be modeled that includes facilities, warehouses, and retailers. Within the model a drop procedure can be employed to eliminate facilities that reach a minimum size and reallocate their demand

among the remaining facilities. Finally, the allocation function can be modified to repeat until there is no improvement in the solution before returning to the location procedure.

3. Conclusion

This paper described an implementation of a modified ALA procedure to incorporate production scale economies into a logistics network design problem. A range of scale economies, modeled using an exponential function, and production-to-transport cost ratios were examined to uncover the impact of these parameters on the algorithm's solutions. The results showed that the model was nearly as efficient as the standard ALA procedure at finding the best solution for a wide range of scale exponents and p/t ratios. These initial findings show promise the approach can offer an alternative to the commonly used piecewise linear methodology used in mixed integer linear programming for modeling nonlinear functions. An extensive list of further work was presented to show the possibilities for extending the modified ALA model to a wide variety of logistic network design problems.

Acknowledgements

This research is supported, in part, by the National Science Foundation under Grant CMS-0229720 (NSF/USDOT).

References

1. de Kok, A.G., Graves, S.C., 2003, "Supply Chain Design and Planning," Handbooks in Operations Research and Management Science, Vol. 11, Supply Chain Management: Design, Coordination and Operation, Chapter 2, pp. 17–93.

2. Shen, Z.J.M., 2005, "A multi-commodity supply chain design problem," IIE Transactions, 37:753–762.

3. Cooper, L., 1963, "Location-allocation problems," Operations Research, 11:331–343.

4. Cooper, L., 1972, “The transportation-location problems,” Operations Research, 20:94–108.

5. Houck, C.R., Joines, J.A., and Kay, M.G., “Comparison of genetic algorithms, random restart, and two-opt switching for solving large location-allocation problems,” Computers & Operations Research, 23(6): 587–596, 1996.

6. Rumelt, R.P., 2001, Note on Strategic Cost Dynamics, POL 2001-1.2, Anderson School at UCLA, pp. 2–3.

7. Kay, M.G., 2006, Matlog: Logistics Engineering Toolbox, North Carolina State University (http://www.ie.ncsu.edu/kay/matlog).

A.1.1 Reprint of Journal Paper: IERC 2007

Title: A Comparison of Meta-Heuristics for Large Scale Facility Location Problems with Economies of Scale

Authors: Michael J. Bucci, Michael G. Kay, Donald P. Warsing
North Carolina State University
Raleigh, NC 27695-7906

Reprint of paper published in proceedings of the 2007 Industrial Engineering Research Conference, IERC, Nashville, Tennessee

A Comparison of Meta-Heuristics for Large Scale Facility Location Problems with Economies of Scale

Michael J. Bucci[*], Michael G. Kay[*], Donald P. Warsing[†]
[*]Fitts Department of Industrial and Systems Engineering
[†]Department of Business Management
North Carolina State University, Raleigh, NC 27695, USA

Abstract

We study facility location-allocation problems that exhibit nonlinear costs in inventory (aggregation savings/risk pooling), production (scale economies), and/or transportation (consolidation). Traditional formulations of this problem either simplify or ignore some of these costs, decompose the problem into several sub-problems, or investigate only small problem instances. We improve and compare several integrative meta-heuristic models that better reflect the complexities of the problem and illustrate how to extend the models to more complex logistics network problems. Computational results are presented for a variety of problems using traditional measurements and a measurement of computational evaluations that provides valuable insight into the efficiency of the heuristics.

Keywords

Location-allocation, heuristics, scale economies

1. Introduction

Facility location-allocation problems involve locating facilities and allocating demand to customers in order to minimize overall system costs given the set of customer locations and demands. These systems typically exhibit nonlinear costs in inventory (aggregation savings/risk pooling), production (scale economies), and/or transportation (consolidation). Traditional formulations of this problem either simplify or ignore some of these costs, decompose the problem into several sub-problems, or investigate small problem instances. We introduce an integrative meta-heuristic procedure, built from earlier work investigating economies of scale in the uncapacitated fixed-charge location problem (UFLP), that better reflects the complexities of the problem and can accommodate large scale problems [1]. We

present computational results for a variety of heuristic approaches over a range of problem sizes. We measure solution speed, solution cost, and the number of computational evaluations in the subroutines. These models are designed to be extended to include inventory aggregation savings and transport consolidation. Additionally, we show our solution technique can be extended to a variety of logistics network design problems.

Facility location models have been widely reported in the literature. Although some problems are modeled and solved with exact methods, heuristics are used in most formulations as these problems are NP hard [2]. The alternate location-allocation (ALA) procedure proposed by Cooper [3] is a widely cited heuristic, with many recent heuristics based on a similar iterative location-allocation principle [2] [4]. A thorough survey and analysis of heuristics for the multi-source Weber problem and *p*-Median problems has been conducted by Brimberg et al. [4]. Their study covers the ALA procedure, tabu search (TS), variable neighborhood search (VNS), fixed neighborhood search (FNS), and *p*-Median heuristic procedures. They observe that relocation methods (facility relocations) are more efficient then neighborhood reallocations (customer reallocations), and the best results are obtained using a VNS search based on a *p*-Median neighborhood structure. Mirchandani et al [5] provides a thorough overview of discrete location models along with the use of discrete models to represent continuous location problems. Daskin [6] reviews well known add/drop/exchange heuristics for the *p*-Median and UFLP. Glover et al [7] explains the use of tabu search techniques, while Hansen et al [2] covers VNS techniques. Resende et al [8] proposes a hybrid multi-start heuristic for the UFLP with tabu search and genetic algorithm procedures.

Whitaker [9] extends the *p*-Median problem to include nonlinear warehousing costs that are continuously concave over a range of possible warehouse sizes. Rumelt [10] proposes a similar cost function for scale economies in manufacturing as a ratio

$$C_1 / C_2 = (S_1 / S_2)^b, \tag{1}$$

where C_1 and C_2 are the unit costs in facilities of size S_1 and S_2, respectively, while b is the scale exponent (estimated to be –0.35 for manufacturing facilities). A similar formula can also be applied to inventory costs. Croxton and Zinn [11] review Maister's work that showed that inventory levels increase as the number of warehouses in the system increases due to the square root law (SRL) of portfolio effect theory. Additionally, they provide a through exposition of the assumptions of the SRL and its applicability to the network design with aggregated product information. Perl and Daskin [12] modeled these nonlinear costs using a fixed cost and linear cost term to simplify the formulation and to reduce computational time, while others have used piecewise linear and discrete point approximations for inventory costs [11] [13].

Adding transportation-based consolidation economies to the facility location problem adds significant complexity. Min et al [14] offers a survey paper of location-routing problem (LRP) methodologies. Liu et al [15] describes a two-phase heuristic, with phase one a route-first followed by a location-allocation procedure. Phase two is an improvement heuristic for the phase one solution. Liu also provides a thorough review of location-routing papers. Melachovsky et al [16] describes a heuristic model for the LRP with nonlinear cost functions that uses a *p*-median approach to get an initial solution and meta-heuristic that combines VNS and TS principles to improve the solution. Although our models are deigned to accommodate transportation consolidation (e.g., via multi-stop routes), we only consider single customer routes in this paper to simplify the analysis of the proposed heuristics.

2. Model Formulation and Implementation

All of the heuristic models were created and executed in MATLAB 7.2, utilizing Matlog [17] as a building block for the models. CPLEX 9.0 was used for the mixed integer formulations of the problems. All tests were run on a 3.2 GHz Intel Pentium 4 computer with 1527 MB of memory running Windows XP. The supply chain model is an extension of a UFLP model previously used by the author [1] that represents a retail distribution network.

The problem includes fixed facility, linear transportation, and nonlinear production costs. Based on results from Brimberg et al [4], which showed that continuous location models required significantly more location cycles than discrete location models, the model was modified to a discrete *p*-Median type of problem in which all retail locations are the possible facility locations. In our largest test this included all 877 three-digit ZIP codes for the continental U.S. with a positive population, where the three-digit ZIP code location is taken to be the population centroid of the constituent five-digit ZIP codes. Smaller tests comprised a subset of the 877-customer problem. As the proposed heuristic models are indifferent to linear or continuous production cost functions, a three segment piecewise linear approximation was used for equation (1) to allow for comparison with CPLEX. The number of facilities to locate was indeterminate at the outset—that is, the heuristic must solve for this parameter. The mathematical formulation for this problem is identical to Whitaker [9] and is not repeated here.

Several models were developed and tested, as summarized in Table 1. The first model is an adaptation of the add/drop heuristics described by Daskin [6] combined with a discrete ALA improvement procedure. In this model, termed A/D-dALA, the greedy ADD construction algorithm described by Daskin [6] is used to add facilities considering fixed facility and transportation costs only to obtain an upper bound on the number of facilities to locate. A greedy DROP procedure is then used to remove facilities from the solution until the total cost is no longer reduced. Our procedure is identical to the procedure offered by Daskin, except our procedure accounts for scale effects on production costs along with the fixed facility and transportation costs. The scale costs, using equation (1), are calculated within each iteration using the facility size from the previous cycle rather then incrementally changing the facility costs with each allocation change. The discrete ALA procedure attempts to improve this initial solution utilizing a Delaunay triangulation (DT) discrete neighborhood relocation search [17]. The allocation procedure within the dALA procedure remains the same as in the greedy ADD procedure—incorporating the fixed facility, transportation, and scale costs in determining the allocation. The second model is an

adaptation of the model proposed by Bucci et al [1] and is a multi-start discrete ALA procedure, termed MS-dALA, that uses the result of an ADD/DROP initialization procedure described above to determine a range for the number of facilities to locate. We determined a range of p–3 to p+1 facilities was sufficient to capture the p value in the best known solution in most problems. At each value of p, forty random starts are performed to improve the likelihood of finding the optimal solution. Although random start procedures have been shown to be increasingly inefficient with larger problem sizes [18], this model is used as a baseline comparison for the effectiveness of the other models.

Table 1: Summary of Heuristic Analyzed

Model	**Initialization method**	**Allocation**	**Location**
1. A/D-dALA	Greedy Add with no EOS, followed by greedy Drop with EOS	Assess all customer allocations in each cycle	FNS using DT
2. MS -dALA	Multi-start with random facility locations	Same as model 1	Same as model 1
3. W	Greedy Add with EOS allocation and location within each Add cycle	Same as model 1	FNS checking all locations in the subgraph
4. W2	Same as model 3	Same as model 1	Same as model 1
5. W-tabu	Same as model 3	Only assess customers on convex hull	Same as model 3, but only assess subgraphs with new facility allocations
6. W2-tabu	Same as model 3	Only assess customers on convex hull	Same as model 1, but only assess subgraphs with new facility allocations

The third model (denoted by W) is a direct implementation of the ADD algorithm described by Whitaker [9], except we continue to add facilities to the solution until there are three solutions that are not better than the best found solution in order to reduce the likelihood the model will terminate prematurely. The Whitaker algorithm is a modification of a greedy

ADD procedure that includes scale costs and performs a location-allocation improvement procedure after each new facility is added to the solution. His model utilizes a relocation procedure that explores all of the customer locations in the subgraph. Whitaker proposes an additional interchange heuristic to improve the ADD procedure solution; however, this was deemed to be too computationally complex for the models we intend to study. The fourth model is a variant of the Whitaker procedure that utilizes the same greedy ADD and allocation procedure, but uses DT in the location procedure. This local fixed neighborhood search is reevaluated after each allocation cycle. The final two models are enhancements to models 3 and 4 that incorporate simple TS methods to reduce the number of computation cycles in the location and allocation phases of the algorithms. Model 5 is an improvement to model 3, while models 6 is an improvement to model 4. The TS feature in the location procedure eliminates all location evaluations in a subgraph or a DT in which the allocation for the subgraph has not changed. This improvement is based on the logical premise that the location of a facility will not change if the allocation has not changed and eliminates unnecessary evaluations. Within the allocation procedure, TS is used to only assess reallocations of the customers on the convex hull of the subgraph, as they are the probable candidates to be reallocated in each allocation cycle.

3. Results and Analysis

To assess the effectiveness of the models, the heuristics were tested on over 20 problems sets with an increasing number of customers (ranging from 4–877 customers). The scale exponent for these tests were –0.1, –0.3, and 1.0, and the fixed facility cost varied from $30,000 to $90,000. The results are discussed below, with a subset of the results (fixed cost = $30,000 and scale exponent = –0.3) presented in Table 2. Several measurements are used to assess the effectiveness of each algorithm. First, total cost relative to the best known solution was analyzed, with a 1% precision level deemed reasonable for these problems based on the efficiency of the MS-dALA procedure. Due to differences in data generation and rounding, identical solutions in CPLEX and the heuristics varied by up to 0.4% in our tests. CPLEX was unable to solve problems with 100 or more customer locations to

optimality (out of memory—OoM). In these cases, where possible we compared to the best found CPLEX solution and the best found CPLEX lower bound. The total solution time for each heuristic includes computation time for initialization procedures. The number of location and allocation evaluations in the "W" heuristics was measured to compare the computational effort of the heuristics and to judge the benefit of the tabu search techniques. In the location procedure, an evaluation is each relocation attempt, while in the allocation procedure an evaluation is each reallocation attempt.

Table 2: Results—Comparison of Heuristics

Model	Customer Locations	**40**	**60**	**100**	**200**	**400**	**877**
1. A/D-dALA	% from best TC	3.5%	0.4%	best	2.0%	1.8%	2.0%
	Time (seconds)	5	3	18	111	1394	11,729
2. MS dALA	% from best TC	best	best	2.3%	2.7%	1.5%	best
	Time (seconds)	51	240	240	7,260	29,753	15,875
3. W	% from best TC	0.9%	0.7%	0.3%	0.5%	best	na
	Time (seconds)	10	29	188	1,884	18,172	OoM
4. W2	% from best TC	1.0%	0.7%	0.3%	0.4%	best	na
	Time (seconds)	16	20	234	1965	20,340	OoM
5. W-tabu	% from best TC	0.9%	0.7%	0.6%	0.7%	0.4%	na
	Time (seconds)	6	11	186	1825	17,547	OoM
6. W2-tabu	% from best TC	0.9%	0.5%	0.3%	best	best	na
	Time (seconds)	14	18	232	3093	19,680	OoM
7. CPLEX	TC	best	best	1.1%	2.2%	na	na
	Time (seconds)	27	1,260	OoM	OoM	OoM	OoM
"Best" heuristic	% from best CPLEX lower bound	0.0%	0.0%	9.8%	23.1%	na	na

Analyzing the total cost relative to the best found solution, the A/D-dALA procedure varied from 0–3.5% while all of the "W" heuristics (models 3–6) were within 1% for all problems. The MS-dALA procedure performed very well for the smaller sized problems, but, as expected, achieved poorer results on larger problems. To assess the ease of finding a good

solution, for each problem, the MS-dALA results for all 40 replications was analyzed to determine the percent of replications that were within 1%, and 5% of the best solution. For smaller problems (<60 customers) 12% of the replications, on average, were within 1% while 90% were within 5%. For larger problems (>80 customers), 0% of the MS-dALA solutions were within 1% of the best solution, while 25% were within 5% of the best solution. This shows both the expected limitations of the multi-start procedure and the effectiveness and importance of the greedy ADD procedures utilized in the "W" heuristics. However, for very large problems (877 customer locations), the "W" heuristics were unable to find a solution (OoM) due to the complexity of these same greedy ADD procedures. A greedy DROP procedure was also considered, but its computational complexity was much higher than the greedy ADD and therefore abandoned.

The computation time in Table 2 provides a good summary of the differences in computation time between the models for the different problems. Since all of the heuristics rely on a greedy ADD procedure, as the problems required solutions with a higher number of facilities, computation time increased significantly. The majority of the computation time in the heuristics appears to be consumed in the greedy ADD step as it evaluates all unused customer locations with the scale costs included. The simplified cost function in the greedy ADD procedure in the A/D-dALA heuristic, in contrast, provides a relatively fast result although it also evaluates all unused customer locations. This is an area that requires further investigation.

The number of computational evaluations was compared for the different heuristics. Table 3 summarizes these results showing the percentage reduction in the number of evaluations in the location and allocation subroutines. Comparing model 4 to 3, the difference being the type of location search used, we observe a minor difference in the total cost of the solution and solution time. However, the DT relocation search in model 4 significantly reduces the number of location evaluations required as the DT relocation is more efficient in exploring the relocation neighborhood. The number of allocation evaluations is not significantly

different, as expected, since the allocation subroutine is identical for both models. The tabu search improvements reduce the number of computational evaluations required in both the location and allocation subroutines. The improvement in the location procedure is due to a realization that many location-allocation cycles do not result in new allocations for all facilities and therefore many areas of the solution space do not need reevaluated in the next iteration of the location subroutine. This result has not been highlighted in other studies and has limited impact on the solution time with relatively simple cost functions. The improvement, however, may be substantial for more complex cost functions. The allocation evaluations were also significantly reduced by limiting the reallocation search to customers along the convex hull of the current allocation.

Table 3: Comparison of the percent reduction in computational evaluations

Model	**Compared to:**	Customer Locations	**60**	**100**	**200**	**400**
4. W2	3. W	Location subroutine	40%	43%	70%	82%
		Allocation subroutine	3%	10%	5%	7%
5. W-tabu	3. W	Location subroutine	50%	69%	55%	86%
		Allocation subroutine	62%	58%	60%	75%
6. W2-tabu	4. W2	Location subroutine	81%	88%	84%	87%
		Allocation subroutine	38%	43%	43%	72%

4. Directions for Future Research

We propose several directions for future research. Tabu search should be expanded to more aspects of models 4, 5, and 6 in an attempt to reduce the computational evaluations further. This would include the complex ADD procedures, which currently consume the majority of the computation time. The less complex greedy ADD heuristic used in the A/D-dALA heuristic provided very good results with significantly less computation time and may provide an alternative initialization method for other heuristics, with TS and VNS potentially improving the efficiency. Additionally, keeping solutions for a range of p values

from the ADD/DROP procedure will improve the probability of finding better solutions when the ADD/DROP formulation does not terminate with the optimal number of facilities to locate. As the current models use only a single neighborhood search, VNS can be added to the models to explore further from the current solution in an attempt to move the solution out of local optima. The multi-start discrete ALA procedure provided reasonable results, but was inefficient for larger problems. Adding VNS and TS to supplement and/or replace the multi-start procedure may make this a more useful method, as these techniques have been successfully implemented in *p*-Median problems [2] [4]. Since the heuristics provide a expandable framework for more complex cost functions,, these models can be extended to incorporate inventory and transportation-based scale costs. Nonlinear inventory costs should be relatively easy to add to the model by simply adding these costs to the current cost function. Transportation economies of scale are planned to be added for vehicle routing, LTL versus TL selection, and/or shipment/route consolidation to extend the applicability of the modeling effort. This will likely include the use of re-allocation methods common to VRP problems such as VRP savings and 2-opt or 3-opt exchange heuristics. The addition of transportation economies and/or other off-line calculations will move the models significantly beyond the current abilities of CPLEX and other mixed integer programming techniques [12]. However, to further the use of CPLEX for comparative purposes with the current model a fixed charge and linear scale cost term can be used in place of the piecewise linear representation of production costs. By significantly reducing the number of integer variables in the problem formulation , this approach allows CPLEX to solve much larger problems to optimality. Initial tests indicate the heuristics exhibit similar behavior with the fixed charge and linear cost term compared to a piecewise linear or continuous cost function; therefore, this change should allow a useful comparison to optimal solutions in much larger problems. Additionally, comparing solutions with CPLEX may lead to a good methodology to select a solution technique (CPLEX, heuristic, or a combination of both) depending on the characteristics of the model.

5. Conclusions

In this paper we have outlined and compared several integrative meta-heuristics for discrete facility location problems that exhibit economies of scale. The problems tested accurately represent retail distribution networks in the continental U.S. The heuristics are extensions of well known techniques used for the multi-source Weber, *p*-median and uncapacitated fixed-charge location problems. The heuristics combined ADD, DROP, ALA, discrete neighborhood search, multi-start, and tabu search techniques. Although the models are intended to investigate problems with continuous nonlinear costs, piecewise linear models were developed to allow comparison to exact solutions using CPLEX—without significantly affecting the behavior of the heuristics. Near optimal solutions were found for a wide range of problems. We have used a measurement of computational evaluations to assess the complexity of the different heuristics. This measure showed tabu search had a significant impact on reducing computational evaluations in the location and allocation subroutines of the models. Testing revealed the initialization procedures to be the most time consuming segments of the heuristics and require further improvement—with tabu search and variable neighborhood search promising improvement techniques. A substantial list of future work was outlined to further analyze the models and extend the models to more complex logistic network problems.

Acknowledgements

This research is supported, in part, by the National Science Foundation under Grant CMS-0229720 (NSF/USDOT).

References

1. Bucci, M.J., Kay, M.G., 2006 “A Modified ALA Procedure for Logistic Network Designs with Scale Economies,” Industrial Engineering Research Conference, IERC 2006, Orlando Florida.

2. Hansen, P., Mladenovic, N., 1997, “Variable Neighborhood Search for the P-median,” Location Science, 5, 207–226.

3. Cooper, L., 1963, “Location-allocation problems,” Operations Research, 11, 331–343.

4. Brimberg, J.,Hansen, P., Mladenovic, N.,Taillard, E.D. (2000) ‘Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem’. *Operations Research*, 48, 444–460.

5. Mirchandani, P.B., Francis, R.L., 1990, Discrete Location Theory, Wiley, New York.

6. Daskin, M.S.,1995, Network and discrete location: models, algorithms, and applications, John Wiley and Sons, New York.

7. Glover, F., Laguna, M., 1997, Tabu Search, Kluwer Academic Publishers, Boston, MA.

8. Resende, M.G.C., Werneck, R.F., 2006, “A hybrid multistart heuristic for the uncapacitated facility location problem,” European Journal of Operational Research, 174, 54–68.

9. Whitaker, R.A. 1985 “Some Add-Drop and Drop-Add Interchange Heuristics for Non-Linear Warehouse Location,” The Journal of the Operational Research Society, 36, 61–70.

10. Rumelt, R.P., 2001, Note on Strategic Cost Dynamics, POL 2001-1.2, Anderson School at UCLA, pp. 2–3.

11. Croxton, K.L., Zinn,W., 2005, “Inventory Considerations in Network Design,” Journal of Business Logistics, 26, 149-168.

12. Perl, J., Daskin, M.S., 1985, “A Warehouse Location-Routing Problem,” Transportation Research Part B, 19, 381–396.

13. Nozick, L.K., Turnquist, M.A., 1998, “Integrating Inventory Impacts into a Fixed-Charge Model for Locating Distribution Centers,” Transportation Research Part E, 34, 173–186.

14. Min, H.M., Jayaraman, V., Srivastava, R., 1998, “Combined location-routing problems: A synthesis and future research directions,” European Journal of Operational Research, 108, 1–15.

15. Liu, S.C., Lee, S.C., 2003, “A two-phase heuristic method for the multi-depot location routing problem taking inventory control decision into considerations,” International Journal of Advanced Manufacturing Technology, 22, 941–950.

16. Melachovsky, J., Prins, C., Calvo, R.W., 2005, “A Metaheuristic to Solve a Location-Routing Problem with Non-Linear Costs,” 11, 375–391.

17. Kay, M.G., 2006, Matlog: Logistics Engineering Toolbox, North Carolina State University (http://www.ie.ncsu.edu/kay/matlog).

18. Hansen, P., Mladenovic, N., Taillard, E., 1998, “Heuristic solution of the multisource Weber problem as a p-median problem,” Operations Research Letters, 22, 55–62.

A.1.2 Additional data not included in journal paper

The following table provides a visual comparison of the methods used in each of the "W" heuristics.

Table 1 Characteristics of "W" heuristics

Heuristic	Construction / Add Locations		Customer Allocation		Facility Location		
	All	Subset	All	Only convex hull	All	Delaunay triangulation	Search only subgraphs with allocation change
3. W1	X		X		X		
4. W2	X		X			X	
5. W1-LAt	X			X		X	X
6. W2- Lt	X					X	X
7. W2-LAt	X			X		X	X
8. W1-S		X			X		
9. W2-S-LAt		X				X	X

To assess the statistical difference in cost between the heuristics we compare the distribution of the %AB cost measurement for the thirty total tests with $n = 100$ and $b \in 0, -0.35, -0.5$ using a non-parametric Wilcoxon rank-sum test [Siegel 1994]. These results, shown in Table 2 are consistent with the general results observable in Table 7 shown in Chapter 3.

Table 2: Wilcoxon Rank-Sum Test Comparison of the %AB measurement

	5. W1-LAt	8. W1-S	4. W2	6. W2-Lt	7. W2-LAt	9. W2-S-LAt	2. MS - dALA	1. AD-dALA
3. W1	1.69	0.61	1.73	3.26	3.64	3.39	2.72	5.49
5. W1-LAt		-0.95	0.27	1.97	2.54	2.37	2.03	5.46
8. W1-S			1.10	2.55	3.02	2.88	2.45	5.38
4. W2				1.23	1.99	1.83	1.82	5.04
6. W2- Lt					0.48	0.42	1.20	4.98
7. W2-LAt						-0.01	1.01	4.85
9. W2-S-LAt							1.04	4.78
2. MS –dALA								2.30

This Wilcoxon rank-sum test compares two of the unpaired samples by sorting and ranking the 60 tests (thirty from each heuristic), and then using a hypothesis test to test whether the two samples come from the same distribution. A test statistic is computed and compared to a two-sided t-test to determine if the samples are significantly different with H_0: *the two samples come from the same distribution*.

We also consider the improvement in the objective function from the start of the heuristic to its final solution with the purpose of determining if we could terminate the heuristics after a certain number of cycles or after a certain percentage improvement without a significant degradation in the total cost. Figure 1 shows the progression of the %AB measurement after a construction, allocation, or location iteration for several of the heuristics. The multi-start and add/drop heuristics start with a better solution since both are initialized with a number of facilities that is close to the number of facilities in the final solution. In contrast, the AD-dALA shows a steep initial improvement as the drop procedure removes facilities from the solution. The improvement is then more gradual as the discrete ALA improvement procedure is utilized. Using only the discrete ALA improvement procedure the MS-dALA procedure shows an erratic improvement trend that typically terminates at a poor solution; however, 160 multi-starts runs compensate for this shortcoming. Using a construction

procedure integrated with the discrete ALA improvement procedure, the W heuristics show a repeating cycle of steep improvement followed by a smaller improvement. The increase in the %AB measurement at the end of W1 and W2-Sw-LAT curves is due to the extension of the heuristics to three non-improving solutions to reduce the likelihood of a premature termination of the procedure.

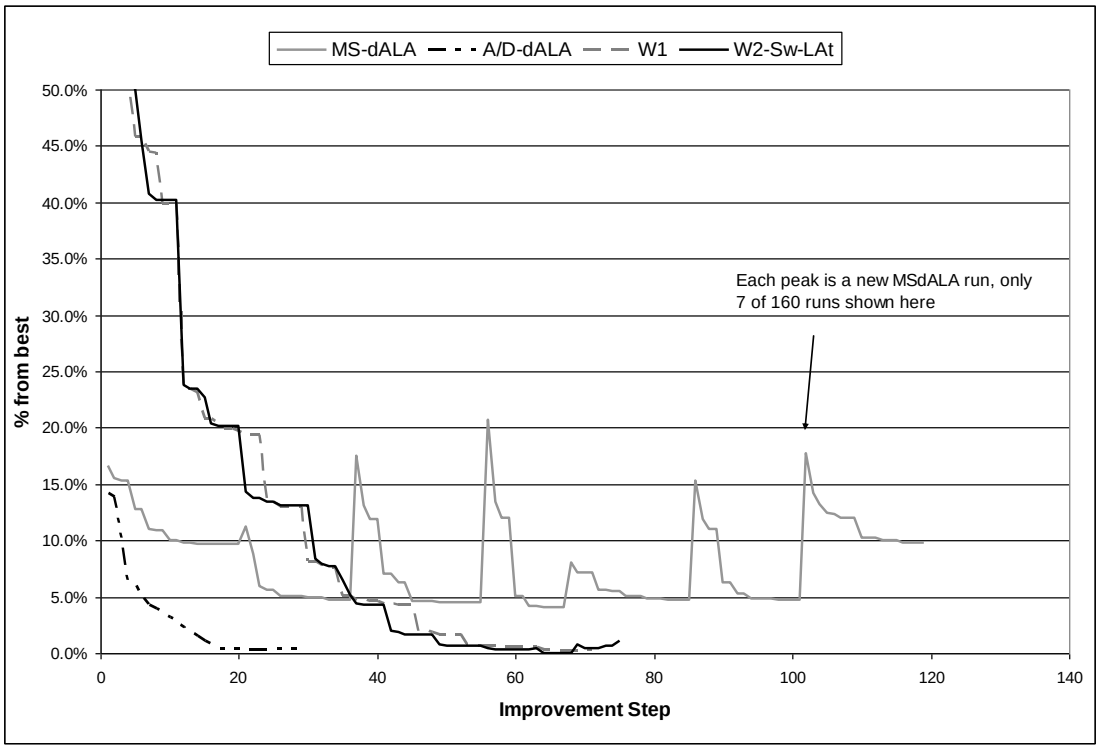

Figure 1: Progression of total cost improvement for $n = 100$, $b = -0.35$

We also look at how easy it is to find a "good" solution. We use the ALA procedure as a baseline. We see in the following Table that the ALA procedure does quite poorly relative to the best solution found. Based on this we deem the problems sufficiently challenges to find "good" solutions.

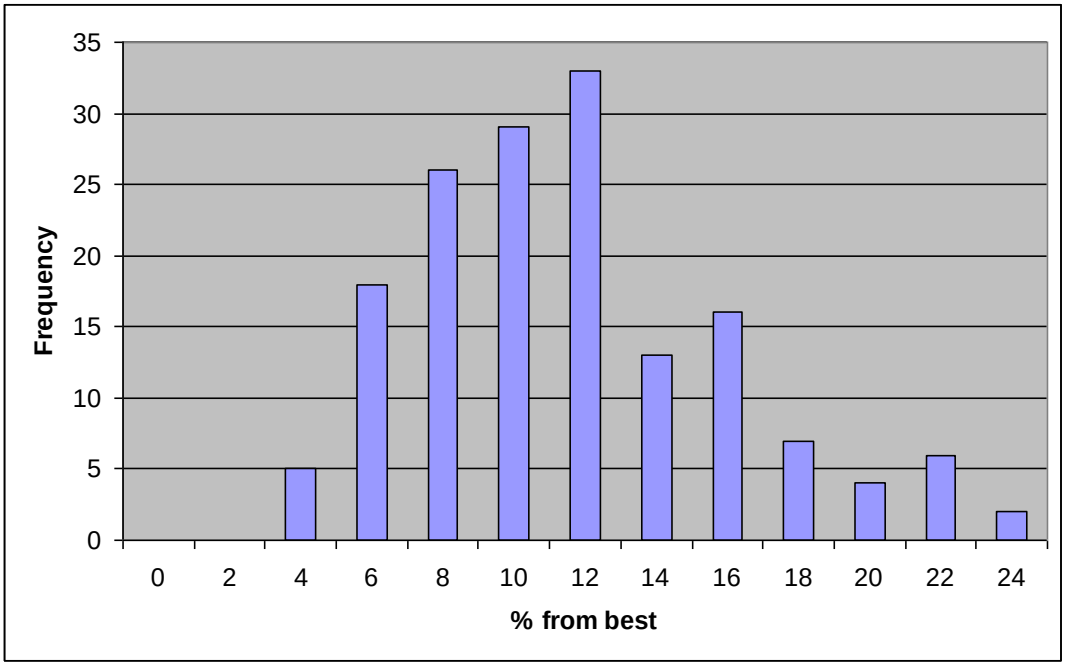

Figure 2: MS-dALA results for 160 multi-start runs with n=100 and b= –0.35

A final analysis was undertaken to investigate the use of customer clustering to reduce the problem size and gain insight into the number of facilities in the final solution. This observation was a byproduct of the main analysis. Since the smaller problems were generated with a weighted random permutation of the n = 877 customer problem with the total demand normalized, these problems are created similarly to a clustered customer problem [Ballou 1994]. The main difference is that these problems do not shift the customer location to a neutral customer location depending on the cluster makeup, rather they use the random weighted permutation as a proxy for the cluster customer location. For $n \in \{60, 100, 200, 400, 600, 877\}$ and b = –0.35 we observe that the range in number of facilities in the final solution varied between 10 – 12 facilities regardless of the size of n.

Not knowing the number of facilities in the solution in advance makes it difficult to reduce the search space in the early stages (see Figure 12) of the heuristics. If the number of facilities in the final solution were known in advance it may be possible to combine some of

the heuristic procedures to reduce the evaluation needed to reach a solution. For example, we may be able to use the AD-dALA procedure in the early iterations and then move toward the more rigorous logic used in the W heuristics as the solution approaches the known facility level. Another approach may be to use a clustering technique [Ballau 1994, Hansen 1997] to generate initial solutions on a reduced problem size (e.g. $n = 100$), and then expand the customer set to a larger size (e.g. $n = 877$) to reach a final solution. To some extent our random weighted permutation used to generate the smaller test problems serves as a clustering technique.

A.1.3 MATLAB code

The MATLAB functions that were developed for the research in Chapter 2 are shown below.

Dependency Report

- Top Level file
    - Data generation procedure
    - Create Piecewise linear approximation
    - Estimate number of facilities in solution
    - Construction subset generation
    - W1 heuristic
    - W1-S heuristic
    - W1-LAt heuristic
    - W2 heuristic
    - W2-Lt heuristic
    - W2-LAt heuristic
    - CPLEX text file generation
    - W2-S-LAt heuristic

The following functions are the subroutines utilized in the above heuristics:

- Discrete ALA single facility solution
- Discrete ALA multi-facility heuristic
- Initialize model
- ADD construction procedure
- ADD construction procedure with subset of possible facility locations
- Allocation procedure
- Allocation procedure with VNS search improvements
- Location procedure
- Location procedure with tabu search improvements

Top Level file for comparing the heuristics

```
% discrete ALA, Whitaker models, multi-start ALA, and CPLEX text file for
solving the uncapacitated fixed charge facility location problem with
production economies of scale

%     k = n-element fixed cost vector, where k(i) is cost of NF at Site i
%           n= number of possible facility locations
%           k is a column vector
%     C = n x m variable cost matrix,
%         where C(i,j) is the cost of serving EF j from NF i
%           m= number of customer locations
%     y = n-element NF site index vector

%%
clear

% INPUT VARIABLES
Input.FC=0;

% ---------   generate data for analysis   ----------------------------
[Input] = datageneration_100_100b(Input); % 100 customer problem
% ------- additional input variables  ----------------------------------
FC=1;
Input.p='mi';  % dists parameter
% Random Number seed
rand ('state',147999);  % seed value for test 1

% MAP of retail locations
 makemap (Input.XY)
 pplot (Input.XY,'b.');


 % ---------  Economy of scale parameters  ----------
% scalef = Scale Exponent, economy of scale exponent
    % a value of 0 means no scale benefit
    % an increasing negative value increases scale benefit
    % and should result in more greedy behavior
Input.scalef=[-0.3]; %for testing

% product cost to transportation cost ratio
%Input.pcost= [0,0.2,0.4,0.6,0.8,1,2,4,6,8];
%Input.pcost= [0,0.2,0.6,1,2,4,6,8];

Input.scalem=Input.scalef;
Input.maxDCsize=Input.prodvol;

%-------------------- Piece-wise linear conversion ---------
 % conversion of continuous production scale economy function for
 %  MATLAB and CPLEX formulation
```

```
% parameters for piece-wise linear calculations

Input.datapt=100; % # of points on continuous curve to be used to create
plc

% convert to piecewise linear approximation - based on allowed error
    Input.ferror=0.20;  % allowed error in piece-wise linear approx.
    % ferror= maximum deviation of any 1 point in the approximation
compared to the actual curve
    [Input] = piecewise_linear_approx_error(Input); % to use plc

% Allocation cycles counter
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];

%-----------    Input for Hybrid and Whitaker program  ------------------
-----
Input.XY; %Customer Locations

k=ones(1,length(Input.XYf))*Input.FC; %fixed facility charge are all equal

distance=dists(Input.XYf,Input.XY,'mi');
C=distance*Input.TLcost*Input.circuity*Input.TLyear;  % cost matrix

for i=1:size(Input.XY,1)
    Input.C(:,i)=C(:,i)*Input.retwt(i,1);  %add retail wt to n x m
variable cost matrix,
      % where C(i,j) is the cost of serving EF j from NF i
      %includes both retail weight and distance
end
Input.k=k;

% get neighbors for all possible NFs locations
TRI = delaunayn(Input.XYf);    % do this upfront instead of each time
location loop is performed
Input.TRI2=tri2list(TRI);
for i=1:length(Input.TRI2)    % change to positive values
    Input.TRI2(i,2)=-Input.TRI2(i,2);
end
g=length(Input.TRI2);

% add y to x option
for i=1:g
    Input.TRI2(end+1,1)=Input.TRI2(i,2);
    Input.TRI2(end,2)=Input.TRI2(i,1);
end
%%
% ----------- Hybrid procedure - plc --------------------------------
```

```
% Use Add to initialize Hybrid procedure
% Then add eos and use Drop procedure.
% Use this to get ballpark number of new facilities
%tic;

% uses piecewise linear curve
tic;
Input.y=[];
[Input,TC,X] = hybridADDeos_plc(Input);

NFhybrid_plc=Input.XYf(Input.y,:);
TChybrid_plc=TC;
Input.NFnum=length(NFhybrid_plc);
allocationhybrid=X;
pplot (NFhybrid_plc,'go');
la_cycles_hybrid=Input.la_cycles;
time_hybrid=toc;
%
% -----------  AD-dALA  --------------------------------
tic
Input.w=Input.retwt; % weights for EF locations
Input.P=Input.XY; % EF locations
Input.p='mi'; % great circle distance for Dists function
%set initial values of TC and TCmin to infinity
TCala=Inf;
TC3=Inf;
TCmin=Inf;
TCallala=[];

% get neighbors for all possible NFs locations
TRI = delaunayn(Input.XYf);    % do this upfront instead of each time
location loop is performed
Input.TRI2=tri2list(TRI);
for i=1:length(Input.TRI2)    % change to positive values
    Input.TRI2(i,2)=-Input.TRI2(i,2);
end
g=length(Input.TRI2);

% add y to x option
for i=1:g
    Input.TRI2(end+1,1)=Input.TRI2(i,2);
    Input.TRI2(end,2)=Input.TRI2(i,1);
end
Input.w=Input.retwt; % weights for EF locations
startDC=NFhybrid_plc;
retloc=Input.retloc;
p=Input.p;
[Xout,TC,W] = ala_onepass(startDC,Input.w,Input.retloc,Input.p);
```

```
Input.scalem=Input.scalef; %for testing
Input.pcostm=Input.pcost;
eosTC=[];  %may not be needed to initiate location procedure
NFloc=Xout;  % NF locations

% using piecewise linear scale costs , solve with discrete ALA procedure
[XYfin,TCfin,Wfin] = dala_eos_plc(NFloc,Input,W);
TCdala_plc_hybrid=TCfin;
NFdala_plc_hybrid=XYfin;
W_dala_plc_hybrid=Wfin;
time_dala_hybrid=toc;

% --------- Add initialization procedure for Whitaker - plc ------

% Use Add and dALA improvement to determine a subset of the possible NFS
to search
% This is a way of defining neighborhoods for the solution space and
% thereby reducing the number of NFs to evaluate in the construction (ADD)
% procedure.  After some level of improvement, we then increase the
% solution space to all NFs.
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.add_cycles=[0];
Input.construct_cycles=[0];
Input.time=[0];
Input.TCall_Whitaker=[];
%Input.percent=0.50;  % minimum percentage of NFs to have in the subset
%[Input,Whitaker] = Whitaker_ad_plc_Npercent(Input);
[Input,Whitaker] = Whitaker3_ad_tabu(Input);

alloc_cycles_Whitaker_Npercent=Input.alloc_cycles;
loc_cycles_Whitaker_Npercent=Input.loc_cycles;
add_cycles_Whitaker_Npercent=Input.add_cycles;
construct_cycles_Whitaker_Npercent=Input.construct_cycles;
time_Whitaker_Npercent=toc;
%time_Whitaker_add_Npercent=Input.time;
Input.TCall_Whitaker=[];
Input.XYf_2=sort(Input.y);

% ----------- W1 - Whitaker base procedure --------------
% Add/Drop eos procedure with piecewise linear costs
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.add_cycles=[0];
[Input,Whitaker] = Whitaker_ad_plc(Input);

% Get best results from Whitaker program
```

```
for i=1:length(Whitaker.TC_data)
    if Whitaker.TC_data(i) == min(Whitaker.TC_data)
        TC_Whitaker = Whitaker.TC_data(i); % lowest TC
        NF_Whitaker= Whitaker.NFloc_data{i,:}; % final NF
        X_Whitaker = Whitaker.X_data{i,:}; % final allocation
        y_Whitaker= Whitaker.y_data{i,:};
        W_Whitaker=zeros(size(X_Whitaker,1),size(X_Whitaker,2));
        for ix =1:length (NF_Whitaker)
            for jx=1:length(Input.retwt)

W_Whitaker(y_Whitaker(ix),jx)=Input.retwt(jx)*X_Whitaker(y_Whitaker(ix),jx
);
            end
        end
    end
end
alloc_cycles_Whitaker=Input.alloc_cycles;
loc_cycles_Whitaker=Input.loc_cycles;
add_cycles_Whitaker=Input.add_cycles;
time_Whitaker=toc;
% ------W1 -S - Whitaker with subset in construction step  -------

tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.construct_cycles=[0];
Input.construct_cycles_2=[0];
Input.add_cycles=[0];
Input.time=[0];
Input.TCall_Whitaker=[];
[Input,Whitaker] = Whitaker_ad_plc_b(Input);
%
% % Get best results from Whitaker program
 for i=1:length(Whitaker.TC_data)
    if Whitaker.TC_data(i) == min(Whitaker.TC_data)
         TC_Whitaker_b = Whitaker.TC_data(i); % lowest TC
         NF_Whitaker_b= Whitaker.NFloc_data{i,:}; % final NF
         X_Whitaker_b = Whitaker.X_data{i,:}; % final allocation
         y_Whitaker_b= Whitaker.y_data{i,:};
         W_Whitaker_b=zeros(size(X_Whitaker_b,1),size(X_Whitaker_b,2));
         for ix =1:length (NF_Whitaker_b)
             for jx=1:length(Input.retwt)

W_Whitaker_b(y_Whitaker_b(ix),jx)=Input.retwt(jx)*X_Whitaker_b(y_Whitaker_
b(ix),jx);
             end
         end
     end
 end
 alloc_cycles_Whitaker_b=Input.alloc_cycles;
```

```
 loc_cycles_Whitaker_b=Input.loc_cycles;
 add_cycles_Whitaker_b=Input.add_cycles;
 %construct_cycles_Whitaker_b=Input.construct_cycles;
 construct_cycles_2_Whitaker_b=Input.construct_cycles_2;
 time_Whitaker_b=Input.time;
 TCall_Whitaker_b=Input.TCall_Whitaker';
% ----------- W1 - LAt - Whitaker Add/Drop with TABU
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.add_cycles=[0];
Input.change_alloc=[0];
[Input,Whitaker] = Whitaker_ad_plc_tabu(Input);

% Get best results from Whitaker program
for i=1:length(Whitaker.TC_data)
    if Whitaker.TC_data(i) == min(Whitaker.TC_data)
        TC_Whitaker_T = Whitaker.TC_data(i); % lowest TC
        NF_Whitaker_T= Whitaker.NFloc_data{i,:}; % final NF
        X_Whitaker_T = Whitaker.X_data{i,:}; % final allocation
        y_Whitaker_T= Whitaker.y_data{i,:};
        W_Whitaker_T=zeros(size(X_Whitaker_T,1),size(X_Whitaker_T,2));
        for ix =1:length (NF_Whitaker_T)
            for jx=1:length(Input.retwt)

W_Whitaker_T(y_Whitaker_T(ix),jx)=Input.retwt(jx)*X_Whitaker_T(y_Whitaker_
T(ix),jx);
            end
        end
    end
end
alloc_cycles_Whitaker_T=Input.alloc_cycles;
loc_cycles_Whitaker_T=Input.loc_cycles;
add_cycles_Whitaker_T=Input.add_cycles;
time_Whitaker_T=toc;

% ----------- W2 ----------------------------
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];

[Input,Whitaker2] = Whitaker2_ad_plc(Input);

% Get best results from Whitaker program
for i=1:length(Whitaker2.TC_data)
    if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
        TC_Whitaker2 = Whitaker2.TC_data(i); % lowest TC
        NF_Whitaker2= Whitaker2.NFloc_data{i,:}; % final NF
```

```
        X_Whitaker2 = Whitaker2.X_data{i,:}; % final allocation
        y_Whitaker2= Whitaker2.y_data{i,:};
        W_Whitaker2=zeros(size(X_Whitaker2,1),size(X_Whitaker2,2));
        for ix =1:length (NF_Whitaker2)
            for jx=1:length(Input.retwt)

W_Whitaker2(y_Whitaker2(ix),jx)=Input.retwt(jx)*X_Whitaker2(y_Whitaker2(ix
),jx);
            end
        end
    end
end
alloc_cycles_Whitaker2=Input.alloc_cycles;
loc_cycles_Whitaker2=Input.loc_cycles;
time_Whitaker2=toc;

% -----------W2-Lt - Whitaker2 with TABU in location --------------
%  Add/Drop eos procedure with piecewise linear costs
% change the subgraph location search with a Delaunay location search with
% a subgraph search as a second level VNS

tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];
Whitaker2.TC_data=[];
Whitaker2.NFloc_data = [];
Whitaker2.y_data = [];
Whitaker2.X_data = [];

[Input,Whitaker2] = Whitaker2_ad_plc_tabu(Input);

% Get best results from Whitaker program
for i=1:length(Whitaker2.TC_data)
    if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
        TC_Whitaker2_tabu = Whitaker2.TC_data(i); % lowest TC
        NF_Whitaker2_tabu= Whitaker2.NFloc_data{i,:}; % final NF
        X_Whitaker2_tabu = Whitaker2.X_data{i,:}; % final allocation
        y_Whitaker2_tabu= Whitaker2.y_data{i,:};

W_Whitaker2_tabu=zeros(size(X_Whitaker2_tabu,1),size(X_Whitaker2_tabu,2));
        for ix =1:length (NF_Whitaker2_tabu)
            for jx=1:length(Input.retwt)

W_Whitaker2_tabu(y_Whitaker2_tabu(ix),jx)=Input.retwt(jx)*X_Whitaker2_tabu
(y_Whitaker2_tabu(ix),jx);
            end
        end
    end
end
```

```
alloc_cycles_Whitaker2_tabu=Input.alloc_cycles;
loc_cycles_Whitaker2_tabu=Input.loc_cycles;
time_Whitaker2_tabu=toc;

% ----------- W2-Lat - Whitaker2 with TABU in location and allocation  ---
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];
Whitaker2.TC_data=[];
Whitaker2.NFloc_data = [];
Whitaker2.y_data = [];
Whitaker2.X_data = [];

[Input,Whitaker2] = Whitaker2_ad_plc_tabu2(Input);

% Get best results from Whitaker program
for i=1:length(Whitaker2.TC_data)
    if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
        TC_Whitaker2_tabu_2 = Whitaker2.TC_data(i); % lowest TC
        NF_Whitaker2_tabu_2= Whitaker2.NFloc_data{i,:}; % final NF
        X_Whitaker2_tabu_2 = Whitaker2.X_data{i,:}; % final allocation
        y_Whitaker2_tabu_2= Whitaker2.y_data{i,:};

W_Whitaker2_tabu_2=zeros(size(X_Whitaker2_tabu_2,1),size(X_Whitaker2_tabu_
2,2));
        for ix =1:length (NF_Whitaker2_tabu_2)
            for jx=1:length(Input.retwt)

W_Whitaker2_tabu_2(y_Whitaker2_tabu_2(ix),jx)=Input.retwt(jx)*X_Whitaker2_
tabu_2(y_Whitaker2_tabu_2(ix),jx);
            end
        end
    end
end
alloc_cycles_Whitaker2_tabu_2=Input.alloc_cycles;
loc_cycles_Whitaker2_tabu_2=Input.loc_cycles;
time_Whitaker2_tabu_2=toc;

% --------  Create CPLEX text file  -------------------------
[filedone] = textfileforCPLEX(Input);
production_cost=Input.maxDCsize*Input.unitcost;

% ------- W2-S-Lat -  Whitaker2 with TABU in location and allocation and
subset for construction   ----------
tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];
```

```
Input.construct_cycles=[0];
Input.construct_cycles_2=[0];
Input.time=[0];
Input.TCall_Whitaker=[];

Whitaker2.TC_data=[];
Whitaker2.NFloc_data = [];
Whitaker2.y_data = [];
Whitaker2.X_data = [];

%[Input,Whitaker2] = Whitaker3_ad_plc_tabu(Input);
[Input,Whitaker2] = Whitaker3_tabu(Input);
% Get best results from Whitaker program
for i=1:length(Whitaker2.TC_data)
    if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
        TC_Whitaker3_tabu_b = Whitaker2.TC_data(i); % lowest TC
        NF_Whitaker3_tabu_b= Whitaker2.NFloc_data{i,:}; % final NF
        X_Whitaker3_tabu_b = Whitaker2.X_data{i,:}; % final allocation
        y_Whitaker3_tabu_b= Whitaker2.y_data{i,:};

W_Whitaker3_tabu_b=zeros(size(X_Whitaker3_tabu_b,1),size(X_Whitaker3_tabu_
b,2));
        for ix =1:length (NF_Whitaker3_tabu_b)
            for jx=1:length(Input.retwt)

W_Whitaker3_tabu_b(y_Whitaker3_tabu_b(ix),jx)=Input.retwt(jx)*X_Whitaker3_
tabu_b(y_Whitaker3_tabu_b(ix),jx);
            end
        end
    end
end
alloc_cycles_Whitaker3_tabu_b=Input.alloc_cycles;
loc_cycles_Whitaker3_tabu_b=Input.loc_cycles;
construct_cycles_Whitaker3_b=Input.construct_cycles_2;
time_Whitaker3_tabu_b=toc;
time_Whitaker3_tabu_add_b=Input.time;
TCall_Whitaker3_tabu_b=Input.TCall_Whitaker';

% ----------- Multi-start discrete ALA -----------
tic
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];
Input.time=[0];

Input.w=Input.retwt; % weights for EF locations
Input.P=Input.XY; % EF locations
Input.p='mi'; % great circle distance for Dists function

%get range for # of NFs to test
NFstart=size(NF_ADeos_plc,1)-2;
```

```
if NFstart <1
    NFstart=1;
end
NFend=size(NF_ADeos_plc,1)+1;
if NFend>length(Input.XYf)
    NFend=length(Input.XYf);
end

TCala=Inf;
TC3=Inf;
TCmin=Inf;
TCallala=[];
Input.TCall=[];
ALAruns=40; % number of multi-starts
startNF=[];

% get neighbors for all possible NFs locations
TRI = delaunayn(Input.XYf);    % do this upfront instead of each time
location loop is performed
Input.TRI2=tri2list(TRI);
for i=1:length(Input.TRI2)    % change to positive values
    Input.TRI2(i,2)=-Input.TRI2(i,2);
end
g=length(Input.TRI2);

% add y to x option
for i=1:g
    Input.TRI2(end+1,1)=Input.TRI2(i,2);
    Input.TRI2(end,2)=Input.TRI2(i,1);
end

%start loop for number of NFs

for numf=NFstart:NFend  %
  for i=1:ALAruns
    %random selection of NF from list of possible NFs
    rn=randperm(length(Input.XYf)); % does not allow the same location to
be
        % used more than once for a starting location
    for j=1:numf
        startNF(j,:)=Input.XYf(rn(j),:);
        %startNF=NFhybrid % for testing
    end
    startDC=startNF;

    % ALA run with no eos to get initial allocation
    w=Input.w;
    retloc=Input.retloc;
    p=Input.p;
    [Xout,TC,W] = ala_onepass(startDC,w,retloc,p);
```

```
    % ALA runs with economy of scale
    Input.scalem=Input.scalef; %for testing
    Input.pcostm=Input.pcost;
    eosTC=[];
    NFloc=Xout;  % NF locations

   [XYfin,TCfin,Wfin,Input] = dala_eos_plc_2(NFloc,Input,W);

    TCallala(end+1)=TCfin;
    if TCfin < TC3 % check to see if lowest value
         TC3=TCfin;
         X3=XYfin;
         W3=Wfin;
         %a3=a1;
    end
  end

  %check to see if cost has been lowered
  if TCmin==Inf % if first loop provides lowest cost
      Xala=XYfin;
      aala=Wfin;
  end

  if TC3 < TCmin
      TCmin=TC3;
      Xala=X3;
      aala=W3;
  end
end  % end for numf loop

TC_MSdala=TCmin;
NF_MSdala=Xala;
X_MSdala=aala;
% makemap (XY);
% alaplot(NF_MSdala,X_MSdala,Input.P,'Discrete ALA - Best');
time_MSdala=toc;
alloc_cycles_MSdala=Input.alloc_cycles;
loc_cycles_MSdala=Input.loc_cycles;
TCall_MSdala=Input.TCall';
```

Data generation procedure:
File name: Paper2_east 2

```
% Data Generation file for discrete ALA vs. CPLEX for uncapacitated fixed
charge facility location
% (UFCFL) problem with production economies of scale
% Hybrid as described by Danskin - add/drop/exchange
% Created by Michael Bucci, Sept. 2006

%     k = n-element fixed cost vector, where k(i) is cost of NF at Site i
%           n= number of possible facility locations
%           k is a column vector
%     C = n x m variable cost matrix,
%         where C(i,j) is the cost of serving EF j from NF i
%           m= number of customer locations
%     y = n-element NF site index vector

function [Input] = discreteALAvsCPLEX_datageneration_200_200c_rdm(Input);

load ('3digitzip_Paper1_3_07_A'); % raw 3 digit zip data
Input.FC=30000;  % fixed facility cost;

Input.prodvol=sum(Input.proddata(:,6))/2;  % product volume
Input.prodcost=(sum(Input.proddata(:,4)))/size(Input.proddata,1)/10; %
avg. prod. cost
% TL cost per mile
    Input.TLcostpermile=2;
    Input.TLcost=Input.TLcostpermile;
% average length of TL shipment = 485 miles
    % from: Transportation America -2002, pg 65
Input.avgTLmiles=245;  % estimate
%calculate TL shipped per year
    % production cost for the year
       % cost to produce 1 unit in largest plant size = $1
    Input.prodcostyr=Input.prodvol*Input.prodcost;
    Input.unitcost=Input.prodcost;
    Input.TLyear=sum(Input.proddata(:,3));

% secuity factor
Input.circuity=1.2;
Input.XYf=Input.XY; % possible facility locations = all customer locations
Input.XYz=Input.XY; % for ALA problem
XYall=Input.XY; % for testing

% transportation cost for year
TCavg = Input.TLcostpermile*Input.avgTLmiles*Input.TLyear;
% product cost per year
PCavg = Input.prodvol*Input.unitcost;
% ----- SAMPLE DATA SET  ----------------------
%if using less than the complete (877 set of locations)
```

```
%Input.prodvol=sum(Input.proddata(:,6));
XYtest=[];
retwttest=[];
volume=[0];
val=200;

cust_val=wtrandperm(Input.retwt,val);

XYtest=Input.XY(cust_val,:);  % customer set
wt_test=sum(Input.retwt(cust_val)); % total wt of reduced set
retwttest=Input.retwt(cust_val);

volume=sum(Input.retwt(cust_val)*Input.prodvol);
retwt1=Input.retwt;  % for testing
Input.retwt=retwttest;


Input.XYf=XYtest;
totwt=sum(Input.retwt);
for i=1:length(Input.retwt)
    Input.retwt(i)=Input.retwt(i)/totwt; %normalized weight
end
Input.retloc=XYtest;
Input.prodvol=volume;
Input.XY=XYtest;
Input.XYf=XYtest;
Input.prodcostyr=Input.prodvol*Input.prodcost;

% transportation cost for year
TCavg_sample = Input.TLcostpermile*Input.avgTLmiles*Input.TLyear;
% product cost per year
PCavg_sample = Input.prodvol*Input.unitcost;
```

Create piecewise linear approximation
File name: piecewise_linear_approx_error

```
% Oct. 2006
% for MATLAB and CPLEX formulation
    % parameters for piece-wise linear calculations

% fs= facility size
% fc= total facility cost

function [Input] = piecewise_linear_approx_error(Input);

% convert non-linear curve into a set of data points
tnlcost=[];
tplcost=[];
tplcost2=[];
tplcost3=[];
nlc=[];

%datapt=100; % determine values at this # of points on continuous curve
nlc(1)=0;
tnlcost(1)=0;
fsize(1)=0;
datarange=Input.datapt;
for i=1:datarange
   pt=i/100;
   %non-linear costs

nlc(i+1)=Input.unitcost*((Input.maxDCsize*pt/Input.maxDCsize)^Input.scalem
); %cost per unit
   tnlcost(i+1)=nlc(i+1)*Input.maxDCsize*pt; %total cost for that location
on curve
   fsize(i+1)=Input.maxDCsize*pt; %
end

% create piece-wise curve
plc(1)=nlc(1);
tplcost(1)=tnlcost(1);

% ---------- Routine to create piece-wise linear curve that will
% create segments based upon size of error allowed

% determine breakpoints by checking deviation values and continue to
% lengthen segments until maximum error is reached. Then start next
segment
% and continue until all breakpoints are determined.

istart=1;
i=3;
```

```
done3=0;
endpt=0;
bpcount=1;
Input.bp(bpcount)=tplcost(1);
bpfsize(bpcount)=0;
tplcost2(1)=0;
while ~done3
%for i=1:datapt
    %tplcost(i)=tplcost(i);
    done4=0;
    while ~done4
        for j=istart+1:i % get piece-wise linear values
            if j==i
            tplcost2(j)=tnlcost(j); % plc costs
            else
            tplcost2(j)=tnlcost(istart)*(i-j)/(i-istart)+tnlcost(i)*(j-
istart)/(i-istart);
            end
        end
        % check values against actual.  If all within acceptable error,
try
        % larger segment.  Stop when no longer working
        OK1=0;
        for k=istart+1:i
            if abs((tplcost2(k)-tnlcost(k))/tnlcost(k))<Input.ferror
            else
                OK1=1;
            end
        end
        i=i+1;

        if OK1==0 % keep extending piecewise linear segment
            tplcost3=tplcost2;  %remember values
            if i==length(tnlcost)
                endpt=1; % at last point on curve, exit loops
                done4=1;
            end
        else % stop
            done4=1;
        end
    end  % end of done4 loop

    % update tplcost and start next segment
    if endpt==1
        Input.bp(bpcount)=tnlcost(end); % reached end of curve...
        bpfsize(bpcount)=fsize(end);
            done3=1;
    else
        if istart+3==i
            tplcost(istart+1)=tnlcost(istart+1);
            Input.bp(bpcount)=tplcost(istart+1); % next breakpoint
```

```
            bpfsize(bpcount)=fsize(istart+1);
            bpcount=bpcount+1; % index counter

            istart=i-2;
            i=istart+2;
        else
            tplcost(istart+1:i-3)=tplcost3(istart+1:i-3);
            tplcost(i-2)=tnlcost(i-2);
            Input.bp(bpcount)=tnlcost(i-2); % next breakpoint
            bpfsize(bpcount)=fsize(i-2);
            bpcount=bpcount+1; % index counter

            istart=i-2;
            i=istart+2;
        end
    end
    if i==length(tnlcost)
        if endpt==0
            done4=1;
            done3=1;
            Input.bp(bpcount)=tnlcost(end);
            bpfsize(bpcount)=fsize(end);
        end
    end
end % entire plc is updated, exit done3 loop


      for i=1:length(Input.bp)
        Input.fc(i+1)=Input.bp(i); % facility cost
        Input.fs(i+1)=bpfsize(i); % facility size

Input.eosplc(i)=Input.unitcost*((Input.fs(i)/Input.maxDCsize)^Input.scalem
);
    end
    Input.fs(1)=0;
    Input.fc(1)=0; % makes the unit cost of size 0 = 0
```

Estimate number of facilities in solution:
File name: hybridADDoes

```
function [y,TC,X] =
hybridADDeos_plc(k,C,y,retwt,prodvol,scalef,unitcost,XYf)

% Hybrid construction procedure for uncapacitated facility location.
% Developed by Mike Bucci April 2006
% used for production economies of scale
% Add facilities without economies of scale, then add economies of scale
% and drop facilities until solution no longer improves
% this gives some general bounds for number of facilities in the optimal
final solution

[n,m] = size(C);
k = k(:);

% Initialize - run ADD with no economies of scale
y=[];
[y,TC,X] = UFCFLadd(k,C,y);
TCadd1=TC;
yadd1=y;
y; %for testing show value
%Xadd1=X;  %allocation (0 or 1 values)

% run DROP with eos to get close to the correct number of NFs

[y,TC,X] = UFCFLdropeos(k,C,y,X,retwt,prodvol,scalef,unitcost,XYf);  % run
DROP program
        y_drop=y;
        TC_drop=TC;
        X_drop=X;

TCbest=TC;
NFtemp=y;
done=0;

y = sort(y);  % final locations of NF
X = logical(sparse(y(argmin(C(y,:),1)),1:m,1,n,m)); %final allocation
    NFspot=y';
    %X=full(X);
    for i=1:length(y)
        NFfin(i,:)=XYf(y(i),:);
    end
    pplot (NFfin,'rx');

TC=TCbest;  % final Total cost
```

Construction subset generation:
File name: UFCFLadd_ss

```
function [Input] = UFCFLadd_ss(Input)
%UFLADD Add construction procedure for uncapacitated facility location.
% Developed by Michael Kay.  Modified by Mike Bucci April 2006

% This algorithm uses Daskin add procedure to determine a subset of
% facilities out of the total number of facilities to use in the Whitaker
% procedures.

% Input Error Checking
******************************************************
% error(nargchk(2,3,nargin));

[n,m] = size(Input.C);
Input.k = Input.k(:);

if nargin < 3, Input.y = []; end
if isempty(Input.y)
   [TC,Input.y] = min(sum(Input.C,2) + Input.k);      % Determine first NF
location
        % min (sum rows and add cost of adding a facility)
else
   TC = sum(Input.k(Input.y)) + sum(min(Input.C(Input.y,:),[],1)); % if
existing locations are provided
end
ny = 1:n;  % n= # of potential NF locations
ny(Input.y) = [];

done = 0;
TC=Inf;

% stopping criteria for subset
NFnum=round(length(Input.XYf)*Input.percent);

while ~done
   TC1 = Inf;  % set initial TC to infinity

   for i = 1:length(ny)
      yi = [Input.y ny(i)]; % adds cost of NF to
      TCi = sum(Input.k(yi)) + sum(min(Input.C(yi,:),[],1));% using
current NF locations sum the minimum in each row of y (finds closest
facility) and add the sum of k(y) which
            % is the total NF cost to get a total cost for current state
      if TCi < TC1 % if better solution found
         i1 = i;
         TC1 = TCi;
      end
```

```
    end

    % add the next best facility to solution
    Input.y = [Input.y ny(i1)]; % set the NF locations to
    ny(i1) = [];
    TC = TC1;

    % check to see if we stop adding facilities
    if length(Input.y)==NFnum
         done = 1;  % stop loop if total cost is not lower
    end
end

Input.y = sort(Input.y);  % final locations of NF
X = logical(sparse(Input.y(argmin(Input.C(Input.y,:),1)),1:m,1,n,m));
```

W1 and W1-S heuristic:

File name: Whitaker_ad_plc
File name: Whitaker_ad_plc_b

```
function[Input,Whitaker] = Whitaker_ad_plc(Input); % W1
function[Input,Whitaker] = Whitaker_ad_plc_b(Input); % W1-S

% Hybrid construction procedure for uncapacitated facility location with
    % eos based on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, November 2006

% Add a new facility each iteration until no improvement

% STEP 0: Find single facility the minimizes total distribution costs of
% the network (transportation, fixed, and warehousing costs)

% STEP 1: To add a facility... For each candidate median (each DC site not
in
% current solution) place a NF at the median location and
% allocate demand based on transportation cost only. Then add fixed
% and production costs. Calculate total cost of this solution.
% Choose the solution that has lowest total cost.

% STEP 2: reallocate customers based on transportation, fixed, and scale
costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement

% STEP 3: relocate DCs based on allocation - possible candidates for NF
are
% all facilities currently within the subgraph(within the allocation)

% Repeat step 2 and 3 until no improvement
% Return to step 1 and continue iteration until no improvement

% ------------------------------------------------------
% STEP 0
% Initialize - run ADD with no economies of scale to get first facility
%  Transportation and fixed facility costs are considered (since scale
%  equation is the same for all facilities can ignore).... then add scale
%  costs to get total cost for this solution
[Input,TC_one_DC,X] = Whitaker_initialize(Input);
Input.time = etime(time2, time1);
Input.TCall_Whitaker(1)=TC_one_DC;
% Keep data on best solution for each NF level (# of facilities)
nNF=1;

Whitaker.TC_data(nNF) = TC_one_DC; % add intitial cost to cost data
```

```
Whitaker.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
Whitaker.X_data{nNF,1} = X; % allocation data
Whitaker.y_data{nNF,1} = Input.y; % y values
nNF=nNF+1;
Xa=X;

% ------------------------------------------------------
% Loop for Steps 1,2,3
done5 = 0;
extend=0;
while ~done5

    % ------------------------------------------------------
    % STEP 1
    % To add a facility... For each candidate median (each DC site not in
    % current solution) place a NF at the median location and
    % allocate demand based on transportation cost only. Then add fixed
    % and production costs. Calculate total cost of this solution.
    % Choose the solution that has lowest total cost.
    [Input,X,TC,W] = Whitaker_add(Input); % W1 heuristic
    [Input,X,TC,W] = Whitaker_add_b(Input); % W1-S heuristic

    Input.time = Input.time + etime(time2, time1);
    Input.TCall_Whitaker(end+1)=TC;
    Input.y;

    % ------------------------------------------------
    % Loops for STEP 2,3
    alloc_loops=0;
    loc_loops=0;
    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
        loops=1;
        while ~done3 % loop until no improvement in the allocation
            %loops=1;

            % STEP 2: reallocate customers based on transportation, fixed,
and scale costs
            % This step is repeated as the reallocations will change
facility sizes
            % and therefore the production costs - until there is no
improvement
            % Constraint is that each facility must serve at least its own
demand point

            [Input,Wa,Xa,TCa] = Whitaker_allocate_plc(Input,X);
            Input.TCall_Whitaker(end+1)=TC;
            % check for improvement
```

```
            if TC > TCa
                TC = TCa;
                W=Wa;
                X=Xa;
                loops=2;
            else
            done3 = 1;
            end
        alloc_loops=alloc_loops+1;
        end % allocation loop

        % STEP 3: relocate DCs based on allocation - possible candidates
for NF are
        % all facilities currently within the subgraph(within the
allocation of the NF)
        TCin=TC;
        [Input,X,W,TC] = Whitaker_loc(Input,X,W,TC);
        loc_loops=loc_loops+1;
        Input.y_last = Input.y;  % for TABU search
        Input.TCall_Whitaker(end+1)=TC;
        if TCin>TC  % change in TC in location solution
                loops=2;
        end
        if loops==1
            done4=1;
        end

    % ------------------------------------------------------

    % Compare best result from Step 1,2,3 iteration with p+1 NFs with best
    % result from previous run with p NFs
    if TC >= min(Whitaker.TC_data)
        done5=1; % stop when no improvement
        extend=extend+1;
            if extend<3% go past the best solution by a few NF to avoid
                 % a premature exit of the program and to have additional
                 % potential solutions to explore
                done5=0;
            end
    end

    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori

    Whitaker.TC_data(nNF) = TC; % Total cost data
    Whitaker.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
    Whitaker.y_data{nNF,1} = Input.y; % y values
    Whitaker.X_data{nNF,1} = X; % allocation data
```

```
    nNF=nNF+1;
    if nNF-1==length(Input.XYf)
        done5=1; % all possible NFs locations have a NF
    end
    Xa=X;
Input.alloc_loops(end+1)=alloc_loops;
Input.loc_loops(end+1)=loc_loops;
end % end for loop for steps 1, 2, and 3
```

W1-LAt heuristic:
File name: Whitaker_ad_plc_tabu

```
function[Input,Whitaker2] = Whitaker_ad_plc_tabu(Input);

[Input,TC_one_DC,X] = Whitaker_initialize(Input);
Input.time = Input.time + etime(time2, time1);

% Keep data on best solution for each NF level (# of facilities)
nNF=1;

Whitaker2.TC_data(nNF) = TC_one_DC; % add intitial cost to cost data
Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
Whitaker2.X_data{nNF,1} = X; % allocation data
Whitaker2.y_data{nNF,1} = Input.y; % y values
nNF=nNF+1;
Xa=X;

% ------------------------------------------------------
% Loop for Steps 1,2,3
extend=0;
done5 = 0;

while ~done5

    % ------------------------------------------------------
    [Input,X,TC,W] = Whitaker_add(Input);
    Input.time = Input.time + etime(time2, time1);

    % for TABU search
    Input.y_last = Input.y;
    Input.W_last=W;
    Input.X_last=X;
    % ------------------------------------------------
    % Loops for STEP 2,3
    % TC = best solution so far
    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
        loops=1;
        while ~done3 % loop until no improvement in the allocation
            % STEP 2: reallocate customers
            [Input,Wa,Xa,TCa] = Whitaker_allocate_plc(Input,X);
            % check for improvement
            if TC > TCa
                TC = TCa;
                W=Wa;
                X=Xa;
                loops=2;
```

```
            else
            done3 = 1;
            end
        end % allocation loop

        % STEP 3: relocate DCs based on allocation
        done_loc=0;
        while ~done_loc
            TCin=TC;
            [Input,X,W,TC] = Whitaker2_loc_tabu(Input,X,W,TC);
            if TCin>TC  % change in TC in location solution
                loops=2;
            else
                done_loc=1;
            end
            Input.TCall_Whitaker(end+1)=min(TC,TCin);
        end
        % For tabu search
        Input.y_last = Input.y;
        Input.W_last=W;
        Input.X_last=X;

        if loops==1
            done4=1;
        end
    end % done4 loop - Repeat step 2 (allocation) and step 3 (location)
until no improvement
    % ------------------------------------------------------
    % Compare best result from Step 1,2,3 iteration with p+1 NFs with best
    % result from previous run with p NFs
    %if Whitaker_TC_data(nNF) >= min(Whitaker_TC_data)
    if TC >= min(Whitaker2.TC_data)
        done5=1; % stop when no improvement
        extend=extend+1;
            if extend<3 % go past the best solution by a few NF to avoid
                 % a premature exit of the program and to have additional
                 % potential solutions to explore
                done5=0;
            end
    end
    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori
    Whitaker2.TC_data(nNF) = TC; % Total cost data
    Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
    Whitaker2.y_data{nNF,1} = Input.y; % y values
    Whitaker2.X_data{nNF,1} = X; % allocation data

    nNF=nNF+1;
    if nNF-1==length(Input.XYf)
        done5=1; % all possible NFs locations have a NF
```

```
    end
end % end for loop for steps 1, 2, and 3
```

W2, W2-Lt, and W2-LAt heuristics

File name: Whitaker2_ad_plc
File name: Whitaker2_ad_plc_tabu
File name: Whitaker3_ad_plc_tabu

```
function[Input,Whitaker2] = Whitaker2_ad_plc(Input); % W2 heuristic
function[Input,Whitaker2] = Whitaker2_ad_plc_tabu(Input); % W2-Lt
heuristic
function[Input,Whitaker2] = Whitaker3_ad_plc_tabu(Input); % W2-LAt
heuristic


[Input,TC_one_DC,X] = Whitaker_initialize(Input);
Input.time = Input.time + etime(time2, time1);

% Keep data on best solution for each NF level (# of facilities)
nNF=1;
Input.TCall_Whitaker(1)=TC_one_DC;
Whitaker2.TC_data(nNF) = TC_one_DC; % add initial cost to cost data
Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
Whitaker2.X_data{nNF,1} = X; % allocation data
Whitaker2.y_data{nNF,1} = Input.y; % y values
nNF=nNF+1;
Xa=X;

% ------------------------------------------------
% Loop for Steps 1,2,3
extend=0;
done5 = 0;

while ~done5

    % --------------------
    % STEP 1 To add a facility
    [Input,X,TC,W] = Whitaker_add(Input);
    Input.y;
    Input.TCall_Whitaker(end+1)=TC;
    % -------------------
    % Loops for STEP 2,3
    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
        loops=1;
        while ~done3 % loop until no improvement in the allocation

            % STEP 2: reallocate customers
            [Input,Wa,Xa,TCa] = Whitaker_allocate_plc(Input,X); %W2 and
```

```
            % W2-Lt heuristic
            [Input,Wa,Xa,TCa] = Whitaker2b_allocate_tabu_plc(Input,X);
            %W2-LAt heuristic



            % check for improvement
            if TC > TCa
                TC = TCa;
                W=Wa;
                X=Xa;
                loops=2;
            else
            done3 = 1;
            end
            Input.TCall_Whitaker(end+1)=TC;
        end

        % STEP 3: relocate DCs based on allocation
        done_loc=0;
        while ~done_loc
            TCin=TC;
            [Input,X,W,TC] = Whitaker2_loc(Input,X,W,TC); % W2 heuristic
            [Input,X,W,TC] = Whitaker2_loc_tabu(Input,X,W,TC); % W2-Lt
      heuristic and W2-LAt heuristic

            if TCin>TC  % change in TC in location solution
                loops=2;
            else
                done_loc=1;
            end
            Input.TCall_Whitaker(end+1)=min(TC,TCin);
        end
        if loops==1
            done4=1;
        end

    end % done4 loop - Repeat step 2 (allocation) and step 3 (location)
until no improvement
    % ------------------------------------------------

    % Compare best result from Step 1,2,3 iteration with p+1 NFs with best
    % result from previous run with p NFs
    if TC >= min(Whitaker2.TC_data)
        done5=1; % stop when no improvement
        extend=extend+1;
            if extend<3 % go past the best solution by a few NF to avoid
                 % a premature exit of the program and to have additional
                 % potential solutions to explore
                done5=0;
```

```
        end
    end

    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori

    Whitaker2.TC_data(nNF) = TC; % Total cost data
    Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
    Whitaker2.y_data{nNF,1} = Input.y; % y values
    Whitaker2.X_data{nNF,1} = X; % allocation data

    nNF=nNF+1;

    if nNF-1==length(Input.XYf)
        done5=1; % all possible NFs locations have a NF
    end

end % end for loop for steps 1, 2, and 3
```

CPLEX text file generation:

File name: textfileforCPLEX

```
function [filedone] = textfileforCPLEX(Input, Input2)
%  -----------------  CPLEX  -----------------------------------
%     k = n-element fixed cost vector, where k(i) is cost of NF at Site i
%           n= number of possible facility locations
%           k is a column vector
%     C = n x m variable cost matrix,
%         where C(i,j) is the cost of serving EF j from NF i
%           m= number of customer locations
%           n= number of facility locations

Ctest=Input.C;
ktest= Input.k';

% ------  create CPLEX lp file  ---------
% variable usage - 0,1 integer values
    % # = a number value representing the
    %     xy matrix in which x = NF and y = EF
    % x#y# = if facility x serves facility y
    % z# = if a facility is located at NF location (0 or 1)
    % q# = what size is each facility (total volume)
    % s#t# = fraction for piecewise linear calculation
        % s=breakpoints, t=facility #
    % a#b# = constraint on z values
        % a=goes from s(1) to s(n-1)
        % b= goes from 1 to x
    %
[fid,mes] = fopen('test60_60_rdm_2b.lp','w');

fprintf(fid,'\\test60_60_rdm\n');
fprintf(fid,'datarun=%g\n',Input2.run);
fprintf(fid,'\\CPLEX data file, Mike Bucci\n');
fprintf(fid,'Minimize\n');
% ----  objective function
    %transportation cost
fprintf(fid,'\\transportation cost\n');
fprintf(fid,'obj: ');
counter=1;
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
            fprintf(fid,'%gx%gy%g+\n',Ctest(i,j),i,j);
    end
end

    % production scale cost
fprintf(fid,'\\production scale cost\n');
for i=1:size(Ctest,1)
```

```
    for j=1:length(Input.fc)
            %for each facility, add (cost at quantity q)*s#t#
            fprintf(fid,'%gs%gt%g+\n',Input.fc(j),j,i);
    end
end

    % fixed facility cost
fprintf(fid,'\\fixed facility cost\n');
for i=1:size(Ctest,1)
    if i==size(Ctest,1)
        fprintf(fid,'%gz%g\n',ktest(i),i);
    else
        fprintf(fid,'%gz%g+\n',ktest(i),i);
    end
end

% ----  add constraints  ---------------------
fprintf(fid,'\\Constraints\n');
fprintf(fid,'Subject to\n');

% each customer is served by 1 facility
fprintf(fid,'\\each customer served by 1 facility\n');
for j=1:size(Ctest,2)
    fprintf(fid,'c%g:',j);
    for i=1:size(Ctest,1)
        if i==size(Ctest,1)
            fprintf(fid,'x%gy%g\n',i,j);
        else
            fprintf(fid,'x%gy%g+\n',i,j);
        end
    end
    fprintf(fid,'=1\n');
end

%if a facility has customers its value in the objective function =1
fprintf(fid,'\\if a facility satisfies demand, it must be open\n');
ccount=size(Ctest,2)+1;
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
        fprintf(fid,'c%g:',ccount);
        fprintf(fid,'x%gy%g-z%g<=0\n',i,j,i);
        ccount=ccount+1;
    end
end

% piece wise linear constraint
%for each facility q=sum of s#t#*cost(q)
fprintf(fid,'\\piecewise quantity calculation = facility demand\n');
for i=1:size(Ctest,1)
    fprintf(fid,'c%g:',ccount);
    fprintf(fid,'q%g-',i);
```

```
    for j=1:length(Input.fs)
        if j==length(Input.fs)
            fprintf(fid,'%gs%gt%g',Input.fs(j),j,i);
        else
            fprintf(fid,'%gs%gt%g-\n',Input.fs(j),j,i);
        end
    end
     fprintf(fid,'=0\n');
     ccount=ccount+1;
end

%piece wise linear constraint
% integer constraints to use the correct part of the piecewise curve
fprintf(fid,'\\piecewise constraint to use correct part of piecewise
curve\n');
for i=1:size(Ctest,1)
    fprintf(fid,'c%g:',ccount);
    fprintf(fid,'s1t%g-a1b%g<=0\n',i,i);
    ccount=ccount+1;

    for j=2:length(Input.fs)
        fprintf(fid,'c%g:',ccount);
        if j<length(Input.fs)
            fprintf(fid,'s%gt%g-a%gb%g-a%gb%g<=0\n',j,i,j-1,i,j,i);
        else
            fprintf(fid,'s%gt%g-a%gb%g<=0\n',j,i,j-1,i);
        end
        ccount=ccount+1;
    end
end
%piece wise linear constraint
% constraint for a#b# must sum to 1
%   for each facility
fprintf(fid,'\\piecewise constraint to ensure only 1 part of curve is
used\n');
for i=1:size(Ctest,1)
    fprintf(fid,'c%g:',ccount);
    for j=1:length(Input.fs)-1
        if j==length(Input.fs)-1
            fprintf(fid,'a%gb%g=1\n',j,i);
        else
            fprintf(fid,'a%gb%g+\n',j,i);
        end

    end
    ccount=ccount+1;
end
%piece wise linear constraint
% constraint for s#t# must sum to 1
%   for each facility
fprintf(fid,'\\all piecewise variables add to 1 for each facility\n');
```

```
for i=1:size(Ctest,1)
    fprintf(fid,'c%g:',ccount);
    for j=1:length(Input.fs)
        if j==length(Input.fs)
        fprintf(fid,'s%gt%g=1\n',j,i);
        else
        fprintf(fid,'s%gt%g+\n',j,i);
        end
    end
    ccount=ccount+1;
end

% piece wise linear constraint
% piecewise quantity equals total demand satisfied by
% the facility  (new constraint from book due to multiple open facilities)
fprintf(fid,'\\facility demand=facility production\n');
for i=1:size(Ctest,1)
    fprintf(fid,'c%g:\n',ccount);
    fprintf(fid,'q%g-\n',i);
    %fprintf(fid,'z%g*\n',i);
    %fprintf(fid,'q%g-\n',i);
    for j=1:size(Ctest,2)
        if j==size(Ctest,2)
               dum1=Input.prodvol*Input.retwt(j);
                fprintf(fid,'%gx%gy%g\n',dum1,i,j);
        else
                dum1=Input.prodvol*Input.retwt(j);
                fprintf(fid,'%gx%gy%g-\n',dum1,i,j);
        end
    end
     fprintf(fid,'=0\n');
     ccount=ccount+1;
end

% ----  Integer constraints/bounds - 0 or 1  ----------
% all s#t# are >=0    % this is the CPLEX default, no code needed
fprintf(fid,'\\integer constraints\n');
fprintf(fid,'generals\n');
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
        fprintf(fid,'x%gy%g\n',i,j);
    end
end
% piece wise linear bound: a#b#
for i=1:size(Ctest,1)
    for j=1:length(Input.fs)-1
        fprintf(fid,'a%gb%g\n',j,i);
    end
end
% end program
fprintf(fid,'end');
```

```
fclose(fid);
filedone=1;
```

W2-S-LAt heuristic:
File name: Whitaker3_tabu

```
function[Input,Whitaker2] = Whitaker3_tabu(Input);

% Hybrid construction procedure for uncapacitated facility location with
    % eos based on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, November 2006

% Add a new facility each iteration until no improvement

% STEP 0: Find single facility that minimizes total distribution costs of
% the network (transportation, fixed, and warehousing costs)

% STEP 1: To add a facility... For each candidate median (each DC site not
in
% current solution) place a NF at the median location and
% allocate demand based on transportation cost only. Then add fixed
% and production costs. Calculate total cost of this solution.
% Choose the solution that has lowest total cost.

% STEP 2: reallocate customers based on transportation, fixed, and scale
costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement

% STEP 3: relocate DCs based on allocation - possible candidates for NF
are
% all facilities currently within the subgraph(within the allocation)

% Repeat step 2 and 3 until no improvement
% Return to step 1 and continue iteration until no improvement

% ------------------------------------------------------
% STEP 0
% Initialize - run ADD with no economies of scale to get first facility
%  Transportation and fixed facility costs are considered (since scale
%  equation is the same for all facilities can ignore).... then add scale
%  costs to get total cost for this solution
time1=clock;
[Input,TC_one_DC,X] = Whitaker_initialize(Input);
time2=clock;
Input.time = Input.time + etime(time2, time1);

% Keep data on best solution for each NF level (# of facilities)
nNF=1;
Input.TCall_Whitaker(1)=TC_one_DC;
Whitaker2.TC_data(nNF) = TC_one_DC; % add initial cost to cost data
Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
```

```
Whitaker2.X_data{nNF,1} = X; % allocation data
Whitaker2.y_data{nNF,1} = Input.y; % y values
nNF=nNF+1;
Xa=X;

% ------------------------------------------------
% Loop for Steps 1,2,3
extend=0;
done5 = 0;

while ~done5

    % ------------------------------------------------------
    % STEP 1
    % To add a facility... For each candidate median (each DC site not in
    % current solution) place a NF at the median location and
    % allocate demand based on transportation cost only. Then add fixed
    % and production costs. Calculate total cost of this solution.
    % Choose the solution that has lowest total cost.
    time1=clock;
    [Input,X,TC,W] = Whitaker_add_b(Input);
    time2=clock;
    Input.time = Input.time + etime(time2, time1);
    %Whitaker2.TC_data(end+1)=TC;  % keep TC from each iteration
    Input.TCall_Whitaker(end+1)=TC;
    % for TABU search
    Input.y_last = Input.y;
    Input.W_last=W;
    Input.X_last=X;

    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
        loops=1;
        while ~done3 % loop until no improvement in the allocation
            % STEP 2: reallocate customers based on transportation, fixed,
            % and scale costs. This step is repeated as the reallocations
            % will change facility sizes and therefore the production
            % costs - until there is no improvement
            % Constraint is that each facility must serve at least its own
demand point
            [Input,Wa,Xa,TCa] = Whitaker2b_allocate_tabu_plc(Input,X);
            % check for improvement
            if TC > TCa
                TC = TCa;
                W=Wa;
                X=Xa;
                loops=2;
            else
            done3 = 1;
```

```
            end
            Input.TCall_Whitaker(end+1)=TC;
        end % allocation loop

        % STEP 3: relocate DCs based on allocation - possible candidates
for NF are
        % all facilities currently within the subgraph(within the
allocation of the NF)
        done_loc=0;
        while ~done_loc
            TCin=TC;
            [Input,X,W,TC] = Whitaker2_loc_tabu(Input,X,W,TC);
            if TCin>TC  % change in TC in location solution
                loops=2;
                %TC=TCin;
            else
                done_loc=1;
            end

            Input.TCall_Whitaker(end+1)=min(TC,TCin);
        end
        % For tabu search
        Input.y_last = Input.y;
        Input.W_last=W;
        Input.X_last=X;

        if loops==1
            done4=1;
        end

    end % done4 loop - Repeat step 2 (allocation) and step 3 (location)
until no improvement
    % ------------------------------------------------
    % Compare best result from Step 1,2,3 iteration with p+1 NFs with best
    % result from previous run with p NFs
    %if Whitaker_TC_data(nNF) >= min(Whitaker_TC_data)
    if TC >= min(Whitaker2.TC_data)
        done5=1; % stop when no improvement
        extend=extend+1;
            if extend<3 % go past the best solution by a few NF to avoid
                 % a premature exit of the program and to have additional
                 % potential solutions to explore
                done5=0;
            end
    end

    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori
    %Whitaker2.TC_data(nNF) = TC; % Total cost data
    Whitaker2.TC_data(nNF) = TC; % Total cost data
```

```
    %Input.TCall_Whitaker(end+1)= TC; %min(TC,TCin);
    Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
    Whitaker2.y_data{nNF,1} = Input.y; % y values
    Whitaker2.X_data{nNF,1} = X; % allocation data

    nNF=nNF+1;
    if nNF-1==length(Input.XYf_2)
        done5=1; % all possible NFs locations have a NF
    end
end % end for loop for steps 1, 2, and 3
```

Subroutine heuristics

Discrete ALA single facility solution

Filename: ala_onepass

```
function [Xout,TC,W] = ala_onepass(startDC,w,retloc,p);

%ALA Alternate location-allocation procedure.

% Copyright (c) 1994-2006 by Michael G. Kay
% Matlog Version 9 13-Jan-2006 (http://www.ie.ncsu.edu/kay/matlog)

% Modified by Mike Bucci 2006 to only find the single facility solution

P=retloc;
X=startDC;
% Input Error Checking
************************************************
error(nargchk(3,5,nargin))

alloc_h = [];
if nargin < 5, loc_h = []; end
if nargin < 4 || isempty(p), p = 'mi'; end
if isa(P,'function_handle'), loc_h = P; P = []; end
if isa(w,'function_handle'), alloc_h = w; w = []; else w = w(:)'; end

if ~isempty(P) && (size(P,2) ~= size(X,2))
   error('Rows in P must equal length of rows in X0.')
elseif ~isempty(P) && ~isempty(w) && (size(P,1) ~= length(w))
   error('Rows in P must equal length of "w".')
end

*******************************************

%if ~isempty(P), alaplot(X,[],P,'Locate'); end

TC = Inf;
D = dists(X,P,p);
W = sparse(argmin(D,1),1:length(w),w,size(X,1),length(w));
TC = sum(sum(W.*D));
Xout=X;
```

Discrete ALA multi-facility heuristic

Filename: dala_eos_plc_2

```
function [XYfin,TCfin,Wfin,Input] = dala_eos_plc_2(NFloc,Input,W);

% ALA procedure developed by Michael Kay
% Copyright (c) 1994-2006 by Michael G. Kay
% Matlog Version 9 13-Jan-2006 (http://www.ie.ncsu.edu/kay/matlog)

% Modified by Mike Bucci in Jan.2006 to end ALA
% procedure if 1 facility has over 99% of demand....
% Modified by Mike Bucci in June 2006 to make function handle
% economies of scale in the allocation and to handle discrete location
% of facilities in the location procedure

% this model loops location cycle until no improvement
% loops allocation cycle until no improvement
% assumes a piecewise linear curve

% parameter definitions

XYZabc=1;

% Initial Allocation with eos
Input.alloc_count=0;
[NFloc_2,W1,TC1,eosTC,Input] = dala_alloc_discrete_plc_2(NFloc,Input,W);
Input.alloc_count=1;
NFloc;

if size(W1,1) ~= size(NFloc,1)
    error('No. rows in W returned by alloc_h and X not equal.')
end

retloc=Input.retloc;

TC = TC1; W = W1; %initial eos solution
TC2=TC;

% ------------------------------------------------------------
% location - allocation loop until no improvement
done = 0;
while ~done
    TC2=TC;
    % location subroutine - keep looping until no improvement
    done2=0;
    while ~done2
        [NFloc,TCloc,Input] = dala_loc(NFloc,Input,W,eosTC);
        % check for improvement
```

```
        if TC > TCloc
            TC = TCloc;
        else
            done2 = 1;
        end
        Input.TCall(end+1)= TC;
    end
   % Allocation subroutine - keep looping until no improvement
   done3=0;
    while ~done3
        [NFloc_2,W1,TCalloc,eosTC,Input] =
dala_alloc_discrete_plc_2(NFloc,Input,W);
        if TC > TCalloc
           TC = TCalloc;
        else
            done3 = 1;
        end
        Input.TCall(end+1)= TC;
    end
   Input.alloc_count=Input.alloc_count+1;

   % check for improvement if any improvement, do location-allocation loop
again
    if TC2 > TCalloc
        TC = TCalloc; W = W1; NFloc=NFloc_2;
        TCalaeos=TC1;
        if max(sum(W,2)>.99)
            done=1;
        end
    else
        if TC2 > TC
            TC2=TC;
        else
            done=1;
        end
    end
end


% output variables
XYfin=NFloc;
TCfin=TC2;
Wfin=W;
```

Initialize model

Filename: Whitaker_initialize

```
function[Input,TC_one_DC,X] = Whitaker_initialize(Input);

% Code by Mike Bucci, November 2006
% eos based on Whitaker (1985) ADD/DROP formulation.

% STEP 0: Find single facility that minimizes total distribution costs of
% the network (transportation, fixed, and warehousing costs)

Input.y=[];
[n,m] = size(Input.C);
Input.k = Input.k(:);

[TC,Input.y] = min(sum(Input.C,2) + Input.k);% Determine first NF location
TC_one_DC=TC;

Input.la_cycles=Input.la_cycles+1; %allocation count

% add scale cost to get TC for single facility solution
% determine size of each DC
X = logical(sparse(Input.y(argmin(Input.C(Input.y,:),1)),1:m,1,n,m));
%final allocation
X=full(X);
Xfinal=X;
for i =1:size(X,1)
    for j=1:length(Input.retwt)
    wtX(i,j)=Input.retwt(j)*X(i,j);
    end
end
W=wtX;
    %wtX % show for testing
DCsize=sum(wtX'*Input.prodvol)';
    totvol=sum(DCsize);
for xy=1:length(DCsize)  % avoid a divide by zero problem
    if DCsize(xy)< 0.5
        DCsize(xy)=0.5;
    end
end

% add production cost to C matrix
    % production cost per year for that mfg. facility to
    % provide product to that retailer
for t=1:n
      for j=1:length(Input.retwt)
        eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
      end
end
```

```
% update C matrix to include both transportation and production scale
costs
Ceos=eosc+Input.C;

[TC_one_DC,Input.y] = min(sum(Ceos,2) + Input.k);% Determine first NF
location
```

ADD construction procedure, and ADD construction procedure with subset of possible facility locations

Filename: Whitaker_add
Filename: Whitaker_add_b

```
function[Input,X,TC,W] = Whitaker_add(Input);
function[Input,X,TC,W] = Whitaker_add_b(Input);

% Hybrid construction procedure for uncapacitated facility location with
    % eos based on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, November 2006
% STEP 1
% To add a facility... For each candidate median (each DC site not in
% current solution) place a NF at the median location and
% allocate demand based on transportation cost only. Then add fixed
% and production costs. Calculate total cost of this solution.
% Choose the solution that has lowest total cost.

[n,m] = size(Input.C);
TC1=Inf;
TC=TC1;
ny = 1:n; % n= all NF locations

% For all facilities
ny(Input.y) = []; % remove NFs currently in the solution from
consideration

% For subset of facilities
% Input.XYf_2 is the NF subset to choose from based on the Whitaker
% initialization procedure
ny=setdiff(Input.XYf_2,Input.y);  % ny= potential NF locations that are in
the subset, but not in the
% the current solution.
NFadd=[];

for i = 1:length(ny)  % for loop to add (one at a time) each candidate
facility and choose the best addition
  yi = [Input.y ny(i)];

  Input.la_cycles=Input.la_cycles+1; %allocation count
  % Allocate based on distance only
        % find minimum in each row of y (finds closest facility
        % based on distance only)
  X = logical(sparse(yi(argmin(Input.C(yi,:),1)),1:m,1,n,m));

  % now add production economies of scale and fixed costs to
  % the solution
          % determine facility size
```

```
          X=full(X);
          for q =1:size(X,1)
                for j=1:length(Input.retwt)
                wtX(q,j)=Input.retwt(j)*X(q,j);
                end
          end

          % wtX % show for testing
          DCsize=sum(wtX'*Input.prodvol)';
          totvol=sum(DCsize);
            for xy=1:length(DCsize)  % avoid a divide by zero problem
                if DCsize(xy)< 0.5
                    DCsize(xy)=0.5;
                end
            end
% add production costs
    % production cost per year for that mfg. facility to
    % provide product to that retailer
  for t=1:n
      for j=1:length(Input.retwt)
            done6=0;
            p=1;
            while ~done6
                if DCsize(t,1)==0.5
                    eosc(t,j)=
totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1); % very large value for
facility with no demand
                    done6=1;
                else
                    if DCsize(t,1)<Input.fs(p+1)

eosc(t,j)=totvol*Input.retwt(j)*(Input.fc(p)+(Input.fc(p+1)-
Input.fc(p))*(DCsize(t,1)-Input.fs(p))/(Input.fs(p+1)-
Input.fs(p)))/DCsize(t,1);
                        done6=1;
                    end
                end
                if Input.fs(end)-DCsize(t,1) <3   % all demand at one
facility

eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
                    done6=1;
                end
                p=p+1; % index to next segment
           end
      end
  end
  % using current NF locations
  TCi = sum(Input.k(yi)) + sum(min(Input.C(yi,:),[],1))+
sum(sum(eosc.*X));
  if TCi < TC1 % if better solution found
```

```
      i1 = i;
      TC1 = TCi;
      Xbest=X;
   end
   Input.construct_cycles_2=Input.construct_cycles_2+size(Input.C,2);

end % for loop to add another facility (for "i" loop)
% ----------
% counts the number of allocations that need to be evaluated in the
% construction procedure
%Input.construct_cycles=Input.construct_cycles+
size(Input.C,1)*size(Input.C,2);

Input.y = [Input.y ny(i1)]; % add the facility to the current best
   % solution found
TC = TC1;
X=Xbest;
for wi =1:size(X,1)
    for wj=1:length(Input.retwt)
        W(wi,wj)=Input.retwt(wj)*X(wi,wj); % create new W matrix for best
solution
    end
end
```

Allocation procedure – base allocation procedure that attempts to reallocate all customers

Filename: Whitaker_allocate_plc

```
function [Input,Wa,Xa,TCa] = Whitaker_allocate_plc(Input,X);

% Code by Mike Bucci, November 2006
% based on Whitaker (1985) ADD/DROP formulation - STEP 2

% STEP 2: reallocate customers based on transportation, fixed, and scale
costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement

% NFloc = NF locations (does not change in this function)
% W1= allocation weight matrix (0 if NF does not serve EF, otherwise
%   the EF weight
% TC1= Total cost after new allocation
% eosTC= total eos cost (for location subroutine)

%Input.alloc_cycles=Input.alloc_cycles+1; %allocation count

% Allocation subroutine for Whitaker algorithm with economy of scale
NFloc=Input.XYf(Input.y,:);
n=size(NFloc,1);
wtX=[];
for i =1:n
    for j=1:length(Input.retwt)
      wtX(i,j)=Input.retwt(j)*X(Input.y(i),j);
    end
end

% Get size of each DC
    DCsize=sum(wtX'*Input.prodvol)';
    totvol=sum(DCsize);
for xy=1:length(DCsize)  % avoid a divide by zero problem
    if DCsize(xy)< 0.5
        DCsize(xy)=0.5;
    end
end
% Get production cost for each DC based on piece-wise linear scale
economies
eosc=[];
for t=1:n
    for j=1:length(Input.retwt)
        done6=0;
        p=1;
        while ~done6 %if (DCsize(i,1)>= fs(p)) % get correct part of plc
            if DCsize(t,1)<=0.5
```

```
                eosc(t,j)=
totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1); % very large value for
facility with no demand
                done6=1;
            else
                if DCsize(t,1)<Input.fs(p+1)

eosc(t,j)=totvol*Input.retwt(j)*(Input.fc(p)+(Input.fc(p+1)-
Input.fc(p))*(DCsize(t,1)-Input.fs(p))/(Input.fs(p+1)-
Input.fs(p)))/DCsize(t,1);
                    done6=1;
                end
            end
            if Input.fs(end)-DCsize(t,1) <3
                eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
                done6=1;
            end
            p=p+1; % index to next segment
        end
    end
end

% Add fixed facility cost per unit to production scale cost
for t=1:n
    for j=1:length(Input.retwt)
        eosc(t,j)= eosc(t,j) +
Input.k(Input.y(t))*Input.retwt(j)*totvol/DCsize(t,1); % add facility cost
        % NOTE: Facility costs are based on DCsize and are spread out to
        % all retailers allocated to the facility.
    end
end

% Get transportation cost
shipcost=dists(NFloc,Input.retloc,'mi');
shipcost=shipcost*Input.circuity*Input.TLyear*Input.TLcostpermile;
for q=1:n
   for j=1:length(Input.retloc)
      shipcost2(q,j)=shipcost(q,j)*Input.retwt(j);
   end
end

% Add production and transportation cost
TandPcost=eosc+shipcost2;
%TPcost=TPcost';
TandP2cost=zeros(n,length(TandPcost));


% find lowest cost DC to retailer path, set others to 0
% constraint that each NF must serve at least the EF at the same location
% eosTC=[0];
for p=1:length(Input.retloc)
```

```
   for q=1:n
     if TandPcost(q,p)-min(TandPcost(:,p))>0.0001;  %should use is0
command
        TandP2cost(q,p)=0;
     else
        TandP2cost(q,p)=TandPcost(q,p);
        %eosTC=eosTC+eosc(q,p); %needed for location subroutine
     end
     Input.alloc_cycles=Input.alloc_cycles+1;
   end
end

% ensure each NF serves the EF at same location
for p=1:length(Input.retloc)
  for q=1:n
      if p==Input.y(q)
          TandP2cost(:,p)=0; % no other NF serves this EF
          TandP2cost(q,p)=TandPcost(q,p); % the NF servs its EF
      end
  end
end

% set Weight matrix
Wa=zeros(size(Input.XYf,1),length(TandPcost));
Xa=zeros(size(Input.XYf,1),length(TandPcost));
for p=1:length(Input.retloc)
    for q=1:n
        if TandP2cost(q,p)> 0.001;
            Wa(Input.y(q),p)=Input.retwt(p,1); % weight matrix
            Xa(Input.y(q),p)=1; % 0,1 allocation matrix
        end
    end
end

TCa=sum(sum(TandP2cost)); %total cost for solution
NFalloc=NFloc;
Walloc=sum(sum(Wa));
```

Allocation procedure with VNS improvements – only tries to reallocate customers on the convex hull

Filename:

```
function [Input,Wa,Xa,TCa] = Whitaker2b_allocate_tabu_plc(Input,X);

% Allocation subroutine for Whitaker algorithm with economy of scale
NFloc=Input.XYf(Input.y,:);
n=size(NFloc,1);
wtX=[];
for i =1:n
    for j=1:length(Input.retwt)
    wtX(i,j)=Input.retwt(j)*X(Input.y(i),j);
    end
end

% Get size of each DC
    DCsize=sum(wtX'*Input.prodvol)';
    totvol=sum(DCsize);
    for xy=1:length(DCsize)  % avoid a divide by zero problem
            if DCsize(xy)< 0.5
                DCsize(xy)=0.5;
            end
    end

% Get production cost for each DC based on piece-wise linear scale
economies
eosc=[];
for t=1:n
    for j=1:length(Input.retwt)
        done6=0;
        p=1;
        while ~done6 %if (DCsize(i,1)>= fs(p)) % get correct part of plc
            if DCsize(t,1)<=0.5
                eosc(t,j)=
totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1); % very large value for
facility with no demand
                done6=1;
            else
                if DCsize(t,1)<Input.fs(p+1)

eosc(t,j)=totvol*Input.retwt(j)*(Input.fc(p)+(Input.fc(p+1)-
Input.fc(p))*(DCsize(t,1)-Input.fs(p))/(Input.fs(p+1)-
Input.fs(p)))/DCsize(t,1);
                    done6=1;
                end
            end
            if Input.fs(end)-DCsize(t,1) <3
                eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
```

```
                done6=1;
            end
            p=p+1; % index to next segment
        end
    end
end

% Add fixed facility cost per unit to production scale cost
for t=1:n
    for j=1:length(Input.retwt)
        eosc(t,j)= eosc(t,j) +
Input.k(Input.y(t))*Input.retwt(j)*totvol/DCsize(t,1); % add facility cost
        % NOTE: Facility costs are based on DCsize and are spread out to
        % all retailers allocated to the facility.
    end
end

% Get transportation cost
shipcost=dists(NFloc,Input.retloc,'mi');
shipcost=shipcost*Input.circuity*Input.TLyear*Input.TLcostpermile;
for q=1:n
   for j=1:length(Input.retloc)
      shipcost2(q,j)=shipcost(q,j)*Input.retwt(j);
   end
end

% Add production and transportation cost
TandPcost=eosc+shipcost2;
%TPcost=TPcost';
TandP2cost=zeros(n,length(TandPcost));

% only consider facilities on the convex hull for re-allocation
% 1. for each NF get facilities in the allocation
hull_all=[];
hull_EF2=[];

for i=1:length(Input.y)
    EF=[];
    count=1;
    for j=1:length(Input.XYf)
        if Input.X_last(Input.y(i),j)==1
            EF(count,:)= Input.XYf(j,:); % EF in the allocation
            count=count+1;
        end
    end
    % 2. get EF on convex hull
    if length(EF)<3
        hull_EF=EF;
    else
        hull_EF=unique(convhulln(EF));
        for xy=1:length(hull_EF)
```

```
            hull_EF2(end+1,:)= EF(hull_EF(xy),:); %get EF locations
        end
    end
end

% erase any duplicates and NF from list
hull_all=unique(hull_EF2,'rows');
% convert hull locations to EF positions in X matrix
hull_EFloc=[];
for i=1:length(hull_all)
    for j=1:length(Input.retloc)
        if hull_all(i,:)==Input.retloc(j,:)
            hull_EFloc(end+1)=j;
        end
    end
end

% for EF on convex hull find lowest cost DC to retailer path
% for those not on convex hull, do not change allocation
% constraint that each NF must serve at least the EF at the same location
for p=1:length(Input.retloc)
    x_hull=0;
    for px=1:length(hull_EFloc)
        if p==hull_EFloc(px)
            x_hull=1; % EF on convex hull
        end
    end
    for q=1:n
        if x_hull==1 % choose lowest cost NF
             if TandPcost(q,p)-min(TandPcost(:,p))>0.001;  %should use is0
command
                TandP2cost(q,p)=0;
             else
                TandP2cost(q,p)=TandPcost(q,p);
                %eosTC=eosTC+eosc(q,p); %needed for location subroutine
             end
             Input.alloc_cycles=Input.alloc_cycles+1;
        else % leave as is
            if X(Input.y(q),p)==1
                TandP2cost(q,p)=TandPcost(q,p);
            else
                TandP2cost(q,p)=0;
            end
        end
    end
end

% ensure each NF serves the EF at same location
for p=1:length(Input.retloc)
  for q=1:n
      if p==Input.y(q)
```

```
            TandP2cost(:,p)=0; % no other NF serves this EF
            TandP2cost(q,p)=TandPcost(q,p); % the NF servs its EF
        end
    end
end

% set Weight matrix
Wa=zeros(size(Input.XYf,1),length(TandPcost));
Xa=zeros(size(Input.XYf,1),length(TandPcost));
for p=1:length(Input.retloc)
    for q=1:n
        if TandP2cost(q,p)> 0.001;
            Wa(Input.y(q),p)=Input.retwt(p,1); % weight matrix
            Xa(Input.y(q),p)=1; % 0,1 allocation matrix
        end
    end
end

TCa=sum(sum(TandP2cost)); %total cost for solution
NFalloc=NFloc;
Walloc=sum(sum(Wa))
```

Location procedure – attempts to relocate all facilities to all customer locations in the subgraph (allocation)

Filename: Whitaker_loc

```
% Mike Bucci Dec. 2006

function [Input,X,W,TC] = Whitaker_loc(Input,X,W,TC);

% based on Whitaker (1985) ADD/DROP formulation - STEP 3
% STEP 3: relocate DCs based on allocation - possible candidates for NF
are all facilities currently within the subgraph(within the allocation)
%
NFloc=Input.XYf(Input.y,:);
% Transportation multiplier to get actual cost
Tmult=Input.circuity*Input.TLyear*Input.TLcostpermile;

% Local Neighborhood search of all NF locations in the subgraph
ichange=0;
for i=1:length(Input.y)
    % get transportation cost + fixed costs for current NF location
    TFbest=sum(Input.k(Input.y(i))) + Tmult * sum(sum(W(Input.y(i),:) .*
dists(Input.XYf(Input.y(i),:),Input.retloc,Input.p)));
    % try to move NF to all possible NF in the subgraph
    for j=1:length(Input.XYf)
        Wvalue=W(Input.y(i),j);
        if Wvalue>0 % EF is within subgraph
            Input.loc_cycles=Input.loc_cycles+1; % location cycle count
            if TFbest > sum(Input.k(j))+ Tmult * sum(sum(W(Input.y(i),:)
.* dists(Input.XYf(j,:),Input.retloc,Input.p)));
                %TFbest= Input.k(j)+ Tmult * sum(sum(W(Input.y(i),:) .*
dists(Input.XYf(j,:),Input.retloc,Input.p)));
                W(j,:)=W(Input.y(i),:);     % shift weighted allocation to
the new location
                W(Input.y(i),:)=0;          % weighted allocation at "old"
facility is now zero
                X(j,:)=X(Input.y(i),:);     % shift (0,1)allocation to the
new location
                X(Input.y(i),:)=0;          % allocation at "old" facility
is now zero
                Input.y(i)=j;               % move to the new location
                ichange=1;
            end
        end
    end
end

% update total cost if locations have changed
if ichange==1;
    NFloc=Input.XYf(Input.y,:);
```

```
    n=size(NFloc,1);
    wtX=[];
    for i =1:n
        for j=1:length(Input.retwt)
        wtX(i,j)=Input.retwt(j)*X(Input.y(i),j);
        end
    end

    % Get size of each DC
        DCsize=sum(wtX'*Input.prodvol)';
        totvol=sum(DCsize);
        for xy=1:length(DCsize)  % avoid a divide by zero problem
            if DCsize(xy)< 0.5
                DCsize(xy)=0.5;
            end
        end

    % Get production cost for each DC based on piece-wise linear scale
economies
    eosc=[];
    for t=1:n
        for j=1:length(Input.retwt)
            done6=0;
            p=1;
            while ~done6 %if (DCsize(i,1)>= fs(p)) % get correct part of
plc
                if DCsize(t,1)==0.5
                    eosc(t,j)=
totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1); % very large value for
facility with no demand
                    done6=1;
                else
                    if DCsize(t,1)<Input.fs(p+1)

eosc(t,j)=totvol*Input.retwt(j)*(Input.fc(p)+(Input.fc(p+1)-
Input.fc(p))*(DCsize(t,1)-Input.fs(p))/(Input.fs(p+1)-
Input.fs(p)))/DCsize(t,1);
                        done6=1;
                    end
                end
                if Input.fs(end)-DCsize(t,1) <3

eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
                    done6=1;
                end
                p=p+1; % index to next segment
           end
        end
    end

    % Add fixed facility cost per unit to production scale cost
```

```
    for t=1:n
        for j=1:length(Input.retwt)
            eosc(t,j)= eosc(t,j) +
Input.k(t)*Input.retwt(j)*totvol/DCsize(t,1); % add facility cost
            % NOTE: Facility costs are based on DCsize and are spread out
to
            % all retailers allocated to the facility.
        end
    end

    % Get transportation cost
    shipcost=dists(NFloc,Input.retloc,'mi');
    %Input.TLcost*Input.circuity*Input.TLyear
    shipcost=shipcost*Input.circuity*Input.TLyear*Input.TLcostpermile;
    for q=1:n
       for j=1:length(Input.retloc)
          shipcost2(q,j)=shipcost(q,j)*Input.retwt(j);
       end
    end

    % Add production and transportation cost
    TandPcost=eosc+shipcost2;
    %TPcost=TPcost';
    TandP2cost=zeros(n,length(TandPcost));


    % Get total costs for solution by setting all unused paths to 0
    for p=1:length(Input.retloc)
       for q=1:n
         if TandPcost(q,p)-min(TandPcost(:,p))>0.001;  %should use is0
command
            TandP2cost(q,p)=0;
         else
            TandP2cost(q,p)=TandPcost(q,p);
         end
       end
    end

    % set Weight matrix

    W=zeros(size(Input.XYf,1),length(TandPcost));
    X=zeros(size(Input.XYf,1),length(TandPcost));
    for p=1:length(Input.retloc)
        for q=1:n
            if TandP2cost(q,p)> 0.001;
                W(Input.y(q),p)=Input.retwt(p,1); % weight matrix
                X(Input.y(q),p)=1; % 0,1 allocation matrix
            end
        end
    end
```

```
    TC=sum(sum(TandP2cost)); %total cost for solution
end

NFlocation=NFloc;
Wloc=sum(sum(W));
TCloc=TC;  % used to check progression of TC
%  -----------   output   --------------------

%NF final locations
NFloc=Input.XYf(Input.y,:);
% Normalize lon-lat pairs
NFloc = normlonlat(NFloc);
```

Location procedure with tabu search improvements - attempts to relocate all facilities to all customer locations in the subgraph (allocation), and only assesses subgraphs with new facility allocations

Filename: Whitaker2_loc_tabu

```
% Mike Bucci Dec. 2006

function [Input,X,W,TC]= Whitaker2_loc_tabu(Input,X,W,TC);

NFloc=Input.XYf(Input.y,:);
% Transportation multiplier to get acutal cost
Tmult=Input.circuity*Input.TLyear*Input.TLcostpermile;

% Get neighbors
TRI3=Input.TRI2;
count=0;
for i=1:length(Input.TRI2)
    ok=0;
    for j=1:length(Input.y)
        if Input.TRI2(i,1)== Input.y(j)
            ok=1;
            j=length(Input.y);
        end
    end
    if ok==1
    else
        TRI3(i-count,:)=[];
        count=count+1;
    end
end

% delete and NF that are in the current solution from the neighbor list
count=0;
length3=length(TRI3);
TRI3b=TRI3;
for i=1:length3
    ok=0;
    for j=1:length(Input.y)
        if TRI3b(i,2)== Input.y(j)
            ok=1;
            j=length(Input.y);
        end
    end
    if ok==1
        TRI3(i-count,:)=[];
        count=count+1;
    end
```

```
end

% Local Neighborhood search of all NF locations in the k=1 neighborhood
ichange=0;
for i=1:length(Input.y)
    loc_test=0; %tabu variable

    % did allocation change for the NF?  If so, check location of all EF
in neighborhood
    change=1;
    loc_change= isequal(W(Input.y(i),:),Input.W_last(Input.y(i),:));
    if loc_change==1
        change=0;
    end
    if abs(change)>0.0000001
        loc_test=1;  % allocation changed
    end

    % get transportation cost + fixed costs for current NF location
    TFbest=sum(Input.k(Input.y(i))) + Tmult * sum(sum(W(Input.y(i),:) .*
dists(Input.XYf(Input.y(i),:),Input.retloc,Input.p)));
    % try to move NF to all possible NF in the neighborhood
    for j=1:length(TRI3)
        loc_test2=0; % tabu variable
        if Input.y(i)==TRI3(j,1)
            if Input.y(i)~=Input.y_last(i) %facility location has changed
                loc_test2=1;
                Input.y_last(i)=Input.y(i);
            end
            if loc_test==1  || loc_test2==1
                Input.loc_cycles=Input.loc_cycles+1; % location cylce
count
                if TFbest > sum(Input.k(TRI3(j,2)))+ Tmult *
sum(sum(W(Input.y(i),:) .*
dists(Input.XYf(TRI3(j,2),:),Input.retloc,Input.p)))+0.001;
                    W(TRI3(j,2),:)=W(Input.y(i),:);     % shift weighted
allocation to the new location
                    Input.W_last(TRI3(j,2),:)=W(Input.y(i),:);
                    W(Input.y(i),:)=0;          % weighted allocation at
"old" facility is now zero
                    Input.W_last(Input.y(i),:)=0;
                    X(TRI3(j,2),:)=X(Input.y(i),:);     % shift
(0,1)allocation to the new location
                    X(Input.y(i),:)=0;          % allocation at "old"
facility is now zero
                    Input.y(i)=TRI3(j,2);               % move to the new
location
                    ichange=1;
                end
            end
        end
```

```
    end
end

% update total cost if locations have changed
if ichange==1;
    NFloc=Input.XYf(Input.y,:);
    n=size(NFloc,1);
    wtX=[];
    for i =1:n
        for j=1:length(Input.retwt)
        wtX(i,j)=Input.retwt(j)*X(Input.y(i),j);
        end
    end

    % Get size of each DC
        DCsize=sum(wtX'*Input.prodvol)';
        totvol=sum(DCsize);
        for xy=1:length(DCsize)  % avoid a divide by zero problem
            if DCsize(xy)< 0.5
                DCsize(xy)=0.5;
            end
        end

    % Get production cost for each DC based on piece-wise linear scale
economies
    eosc=[];
    for t=1:n
        for j=1:length(Input.retwt)
            done6=0;
            p=1;
            while ~done6 %if (DCsize(i,1)>= fs(p)) % get correct part of
plc
                if DCsize(t,1)==0.5
                    eosc(t,j)=
totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1); % very large value for
facility with no demand
                    done6=1;
                else
                    if DCsize(t,1)<Input.fs(p+1)

eosc(t,j)=totvol*Input.retwt(j)*(Input.fc(p)+(Input.fc(p+1)-
Input.fc(p))*(DCsize(t,1)-Input.fs(p))/(Input.fs(p+1)-
Input.fs(p)))/DCsize(t,1);
                        done6=1;
                    end
                end
                if Input.fs(end)-DCsize(t,1) <3

eosc(t,j)=totvol*Input.retwt(j)*Input.fc(end)/DCsize(t,1);
                    done6=1;
                end
```

```
                p=p+1; % index to next segment
            end
        end
    end

    % Add fixed facility cost per unit to production scale cost
     for t=1:n
        for j=1:length(Input.retwt)
            eosc(t,j)= eosc(t,j) +
Input.k(t)*Input.retwt(j)*totvol/DCsize(t,1); % add facility cost
            % NOTE: Facility costs are based on DCsize and are spread out
to
            % all retailers allocated to the facility.
        end
    end

    % Get transportation cost
    %shipcost2=zeros(n,length(Input.retloc));
    shipcost=dists(NFloc,Input.retloc,'mi');
    %Input.TLcost*Input.circuity*Input.TLyear
    shipcost=shipcost*Input.circuity*Input.TLyear*Input.TLcostpermile;
    for q=1:n
       for j=1:length(Input.retloc)
          shipcost2(q,j)=shipcost(q,j)*Input.retwt(j);
       end
    end

    % Add production and transportation cost
    TandPcost=eosc+shipcost2;
    %TPcost=TPcost';
    TandP2cost=zeros(n,length(TandPcost));

    % Get total costs for solution by setting all unused paths to 0
    for p=1:length(Input.retloc)
       for q=1:n
         if TandPcost(q,p)-min(TandPcost(:,p))>0.001;  %should use is0
command
            TandP2cost(q,p)=0;
         else
            TandP2cost(q,p)=TandPcost(q,p);
         end
       end
    end

    % set Weight matrix

    W=zeros(size(Input.XYf,1),length(TandPcost));
    X=zeros(size(Input.XYf,1),length(TandPcost));
    for p=1:length(Input.retloc)
        for q=1:n
```

```
            if TandP2cost(q,p)> 0.001;
                W(Input.y(q),p)=Input.retwt(p,1); % weight matrix
                X(Input.y(q),p)=1; % 0,1 allocation matrix
            end
        end
    end
    TC=sum(sum(TandP2cost)); %total cost for solution
end

NFlocation=NFloc;
Wloc=sum(sum(W));
TCloc=TC;  % used to check progression of TC
%  -----------   output   --------------------

%NF final locations
NFloc=Input.XYf(Input.y,:);
% Normalize lon-lat pairs
NFloc = normlonlat(NFloc);
```

Appendix B: Supplemental Information for Chapter 3

B.1.1 MATLAB code

The MATLAB functions (code) that were developed for the research in Chapter 3 are shown below.

Dependency Report

- Top Level file
  - Data generation procedure
  - Construction subset generation
  - Meta-heuristic that combines constructive ADD procedure with ALA procedure
    - Initialize model
      - "Square Root Law" calculation – SRL_1
      - General Concave Cost function calculation – SRL_2
      - Explicit calculation for safety stock inventory – SRL_0
    - ADD construction procedure
      - "Square Root Law" calculation – SRL_1
      - General Concave Cost function calculation – SRL_2
      - Explicit calculation for safety stock inventory – SRL_0
    - Allocation procedure
      - "Square Root Law" calculation – SRL_1
      - General Concave Cost function calculation – SRL_2
      - Explicit calculation for safety stock inventory – SRL_0
    - Location procedure
      - "Square Root Law" calculation – SRL_1
      - General Concave Cost function calculation – SRL_2
      - Explicit calculation for safety stock inventory – SRL_0

Top Level file for comparing inventory levels between SRL_0 to SRL_1 and SRL_2 for a common solution:

```
% Heuristics used from Journal Paper 1
    % AD-dALA, MS-dALA, and W2-Sw-LAT

% Created by Michael Bucci, September 2007
clear

rand_num=[
    7.0059
    7.3839
    5.6240
    5.4740
    4.2042
    6.9486
    1.3344
    5.8224
    9.8396
    6.7078
    8.1510
    3.3140
    2.8079
    2.9067
    1.0229
    8.7454
    9.1320
    8.7007
    2.1475
    0.5481
    9.7345
    3.1869
    1.6630
    8.0927
    3.2106
    1.7162
    0.7793
    9.5193
    8.2543
    5.7452];
rand_num=rand_num*1000000;
rand_num=rand_num';
%%
runs=1;
for data_run=1:runs
    data_run % show for testing

    Input2.run=data_run;
    rand ('state',rand_num(data_run)); Input.FC=0;
```

```
    % ---------   generate data for analysis   ---------------------------
-
    [Input] = Paper2_east2_rho_groups(Input);
    Input.percent=1.0;  % minimum percentage of NFs to have in the subset
    %Input.percent=1.0;
    % ------- additional input variables  -------------------------
    Input.XYf=Input.XY;
    %-----------    Input for Hybrid and Whitaker program  ------------
    % get neighbors for all possible NFs locations
    TRI = delaunayn(Input.XYf);
    Input.TRI2=tri2list(TRI);
    for i=1:length(Input.TRI2) % change to positive values
        Input.TRI2(i,2)=-Input.TRI2(i,2);
    end
    g=length(Input.TRI2);
    % add y to x option
    for i=1:g
        Input.TRI2(end+1,1)=Input.TRI2(i,2);
        Input.TRI2(end,2)=Input.TRI2(i,1);
    end
    %%  ---- Daskin Add used to create construction subset -------
    tic;
    Input.y=[];
    Input.time=[0];
    [Input] = UFCFLadd_ss(Input);
    time_Add_ss=toc;
    Input.XYf_2=sort(Input.y);
    %  W2-Sw-LAT-rho  Whitaker2 with TABU in location and allocation and
    % subset in construction procedure with rho for inventory calculation

    tic;
    Input.y=[];
    Input.alloc_cycles=[0];
    Input.loc_cycles=[0];
    Input.la_cycles=[0];
    Input.construct_cycles=[0];
    Input.construct_cycles_2=[0];
    Input.time=[0];
    Input.TCall_Whitaker=[];
    Input.alloc_change=0;
    Input.loc_change=0;
    Whitaker2.TC_data=[];
    Whitaker2.NFloc_data = [];
    Whitaker2.y_data = [];
    Whitaker2.X_data = [];
    Whitaker2.degen=[0];
    Whitaker2.NF_num=[];
    Whitaker2.inv_cost_data=[];
    Whitaker2.trans_cost_data=[];

    Input.SRL=1;
```

```
    [Input,Whitaker2] = Paper2_srl_a(Input, Whitaker2);
% Get best results from Whitaker program
    for i=1:length(Whitaker2.TC_data)
        if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
            TC_W2_SwLAT_rho = Whitaker2.TC_data(i); % lowest TC
            NF_W2_SwLAT_rho= Whitaker2.NFloc_data{i,:}; % final NF
            X_W2_SwLAT_rho = Whitaker2.X_data{i,:}; % final allocation
            y_W2_SwLAT_rho= Whitaker2.y_data{i,:};

W_W2_SwLAT_rho=zeros(size(X_W2_SwLAT_rho,1),size(X_W2_SwLAT_rho,2));
            for p=1:length(Input.retloc)
                for q=1:length(y_W2_SwLAT_rho)
                    if X_W2_SwLAT_rho(y_W2_SwLAT_rho(q),p)==1

W_W2_SwLAT_rho(y_W2_SwLAT_rho(q),p)=Input.retwt(p); % weight matrix
                    end
                end
            end
        end
    end
    % delete any unused faciltiies (degeneracy)
    y_temp=y_W2_SwLAT_rho;
    for g=1:length(y_W2_SwLAT_rho);
                g_back=length(y_W2_SwLAT_rho)+1-g;
                if sum(X_W2_SwLAT_rho(y_W2_SwLAT_rho(g_back),:),2)==0;
                    y_temp(g_back)=[];
                end
    end
    y_W2_SwLAT_rho=y_temp;
    NF_W2_SwLAT_rho= Input.XYf(y_W2_SwLAT_rho,:);

    %pplot (Input.XY,'b.');
    alloc_cycles_W2_SwLAT_rho=Input.alloc_cycles;
    loc_cycles_W2_SwLAT_rho=Input.loc_cycles;
    construct_cycles_W2_SwLAT_rho=Input.construct_cycles_2;
    %time_W2_SwLAT_srl=toc;
    TCall_W2_SwLAT_rho=Input.TCall_Whitaker';

    % Plot final solution
    Wcurrent=W_W2_SwLAT_rho(y_W2_SwLAT_rho,:);
    sum(Wcurrent,2);
    % makemap (Input.XY);
    %alaplot(Input.XYf(y_W2_SwLAT_rho,:),Wcurrent,Input.XY,'No Rho -
Final');
    time_SRL3=toc;

    %% Get transportation and inventory costs for current best solution
    X_input=X_W2_SwLAT_rho;
    Input.y=y_W2_SwLAT_rho;
        %inv_cost=0;
        [inv_cost]= W2_SS_srl3(Input,X_input);
```

```
        inv_cost_no_rho=inv_cost;
        %calculate transportation cost
        trans_cost_no_rho=sum(sum(Input.C.*X_input));
        % calculate fixed facility cost
        fixed_cost_no_rho=sum(Input.k(Input.y));
        % total cost
        TC_no_rho=trans_cost_no_rho+inv_cost_no_rho+fixed_cost_no_rho;
    % SRL1, original formula
        [inv_cost]= W2_SS_srl1(Input,X_input);
        inv_cost_SRL1=inv_cost;
    % SRL2, uses facility size
        %Input.scale_exp=0.5;
        Input.scale_exp=0.5;
        %Input.scale_exp=0.3;
        [inv_cost]= W2_SS_srl2(Input,X_input);
        inv_cost_SRL2=inv_cost;
    % SRL3, explicit calculation with mean and c.v.
    [inv_cost]= W2_SS_srl3(Input,X_input);
        inv_cost_SRL3=inv_cost;

    % Data collection
    Output.TC_NF= Whitaker2.TC_data;
    Output.Inv_cost(data_run,1)=inv_cost_SRL3; % inventory costs
    Output.Inv_cost(data_run,2)=inv_cost_SRL2; % inventory costs
    Output.Inv_cost(data_run,3)=inv_cost_SRL1; % inventory costs
    Output.trans_cost(data_run,1)=trans_cost_no_rho; % transportation cost
    Output.time(data_run,1)= time_SRL3; % time
    Output.NFnum(data_run,1)=length(NF_W2_SwLAT_rho); % # of NFs
    NFsizes=sum(Wcurrent,2);
    for i=1:size(NF_W2_SwLAT_rho,1)
        Output_NFloc(data_run,1,i,1)=NF_W2_SwLAT_rho(i,1);
        Output_NFloc(data_run,1,i,2)=NF_W2_SwLAT_rho(i,2);
        Output_NFsizes(data_run,1,i)=NFsizes(i);
    end

    % save allocation
    Output.X_input(data_run,1,:,:)=X_input; % SRL3 allocation
    Output.X_input=[];

    clear TC* NF* X* a* c* l* t* y* ;
end % end for "for" loop for data runs
```

Top Level file for comparing SRL_0 to SRL_1 and SRL_2 on a total cost, solution time, and solution structure:

```
%Top level heuristic for analysis of safety stock approximations and
including correlation in inter-customer demands
% Heuristics are adaptations from dissertation Chapter 2
    % AD-dALA, MS-dALA, and W2-Sw-LAT

% Created by Michael Bucci, September 2007
clear

rand_num=[
    7.0059
    7.3839
    5.6240
    5.4740
    4.2042
    6.9486
    1.3344
    5.8224
    9.8396
    6.7078
    8.1510
    3.3140
    2.8079
    2.9067
    1.0229
    8.7454
    9.1320
    8.7007
    2.1475
    0.5481
    9.7345
    3.1869
    1.6630
    8.0927
    3.2106
    1.7162
    0.7793
    9.5193
    8.2543
    5.7452];
rand_num=rand_num*1000000;
rand_num=rand_num';
%%
runs= 30; % # of instances (runs)
for data_run=1:runs

Input2.run=data_run;
rand ('state',rand_num(data_run)); Input.FC=0;
```

```
% ---------   generate data for analysis   ---------------------------
[Input] = Paper2_east2(Input);

Input.percent=0.25;  % minimum percentage of NFs to have in the subset
    % generation for the construction procedure
% ------- additional input variables  ----------------------------------
Input.XYf=Input.XY;
%-----------    Input for Hybrid and Whitaker program  ------------------
-----
% get neighbors for all possible NFs locations
TRI = delaunayn(Input.XYf);
Input.TRI2=tri2list(TRI);
for i=1:length(Input.TRI2) % change to positive values
    Input.TRI2(i,2)=-Input.TRI2(i,2);
end
g=length(Input.TRI2);
% add y to x option
for i=1:g
    Input.TRI2(end+1,1)=Input.TRI2(i,2);
    Input.TRI2(end,2)=Input.TRI2(i,1);
end
%%  ---- Daskin (1995) Add procedure used to create construction subset --
-----
tic;
Input.y=[];
Input.time=[0];
[Input] = UFCFLadd_ss(Input);
time_Add_ss=toc;
Input.XYf_2=sort(Input.y);
%% ---------- Run each of the SRL methods using: -----
%  W2-Sw-LAT-rho  Whitaker2 with TABU in location and allocation and
% subset in construction procedure with rho for inventory calculation

for SRL_select=1:3 % solve with each of the three method
    Input.SRL=SRL_select;
    tic;
    Input.y=[];
    Input.alloc_cycles=[0];
    Input.loc_cycles=[0];
    Input.la_cycles=[0];
    Input.construct_cycles=[0];
    Input.construct_cycles_2=[0];
    Input.time=[0];
    Input.TCall_Whitaker=[];
    Input.alloc_change=0;
    Input.loc_change=0;
    Whitaker2.TC_data=[];
    Whitaker2.NFloc_data = [];
    Whitaker2.y_data = [];
    Whitaker2.X_data = [];
```

```
    Whitaker2.degen=[0];
    Whitaker2.NF_num=[];
    Whitaker2.inv_cost_data=[];
    Whitaker2.trans_cost_data=[];

    [Input,Whitaker2] = Paper2_heuristic_a(Input, Whitaker2);

    % Get best results from the heuristic
    for i=1:length(Whitaker2.TC_data)
        if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
            TC_W2_SwLAT_rho = Whitaker2.TC_data(i); % lowest TC
            NF_W2_SwLAT_rho= Whitaker2.NFloc_data{i,:}; % final NF
            X_W2_SwLAT_rho = Whitaker2.X_data{i,:}; % final allocation
            y_W2_SwLAT_rho= Whitaker2.y_data{i,:};

W_W2_SwLAT_rho=zeros(size(X_W2_SwLAT_rho,1),size(X_W2_SwLAT_rho,2));
            for p=1:length(Input.retloc)
                for q=1:length(y_W2_SwLAT_rho)
                    if X_W2_SwLAT_rho(y_W2_SwLAT_rho(q),p)==1

W_W2_SwLAT_rho(y_W2_SwLAT_rho(q),p)=Input.retwt(p); % weight matrix
                    end
                end
            end
        end
    end
     %pplot (Input.XY,'b.');
    alloc_cycles_W2_SwLAT_rho=Input.alloc_cycles;
    loc_cycles_W2_SwLAT_rho=Input.loc_cycles;
    construct_cycles_W2_SwLAT_rho=Input.construct_cycles_2;
    %time_W2_SwLAT_srl=toc;
    TCall_W2_SwLAT_rho=Input.TCall_Whitaker';

    % Plot final solution
        %Wcurrent=W_W2_SwLAT_rho(y_W2_SwLAT_rho,:);
        %sum(Wcurrent,2);
        %alaplot(Input.XYf(y
    time_SRL=toc;

    %% Get transportation and inventory costs for current best solution
    %Input.retvar=Input.retvar'
    X_input=X_W2_SwLAT_rho;
    Input.y=y_W2_SwLAT_rho;
        %inv_cost=0;
        if Input.SRL == 1
            [inv_cost]= W2_SS_srl3(Input,X_input);
        elseif Input.SRL==2
            [inv_cost]= W2_SS_srl2(Input,X_input);
        else
            [inv_cost]= W2_SS_srl1(Input,X_input);
        end
```

```
        inv_cost_SRL=inv_cost;
        %calculate transportation cost
        trans_cost=sum(sum(Input.C.*X_input));
        % calculate fixed facility cost
        fixed_cost=sum(Input.k(Input.y));
        % total cost
        TC=trans_cost+inv_cost_SRL+fixed_cost;

        % get inventory cost with SRL3 for apples to
        % apples comparison
        [inv_cost]= W2_SS_srl3(Input,X_input);
        inv_cost2=inv_cost;
        TC2=trans_cost+inv_cost2+fixed_cost;

    % Data collection

    Output.Inv_cost(data_run,Input.SRL)=inv_cost_SRL; % inventory costs
    Output.Trans_cost(data_run,Input.SRL)=trans_cost; % transportation
cost
    Output.Fixed_cost(data_run,Input.SRL)=fixed_cost; % inventory costs
    Output.Total_cost(data_run,Input.SRL)=TC; % total costs

    Output.Inv_cost2(data_run,Input.SRL)=inv_cost2; % inventory costs
    Output.Total_cost_adjusted(data_run,Input.SRL)=TC2; % adjusted total
costs

    Output.time(data_run,Input.SRL)= time_SRL; % time

    Output.NFnum(data_run,Input.SRL)=length(NF_W2_SwLAT_rho); % # of NFs

    NFsizes=sum(Wcurrent,2);
    for i=1:size(NF_W2_SwLAT_rho,1)
        Output_NFloc(data_run,Input.SRL,i,1)=NF_W2_SwLAT_rho(i,1);
        Output_NFloc(data_run,Input.SRL,i,2)=NF_W2_SwLAT_rho(i,2);
        Output_NFsizes(data_run,Input.SRL,i)=NFsizes(i);
    end

    % save allocation
    Output.X_input(data_run,Input.SRL,:,:)=X_input; % SRL allocation

    clear TC* NF* W* X* a* c* l* t* y* ;
end % end for SRL selection
end % end for "for" loop for data runs

%%  ---------- Data collection

% column 1 = total cost
% column 2= time in seconds
% column 3 = construction cycles
```

```
% column 4 = allocation cycles
% column 5 = location cycles
% column 6 = NF locations
% column 7 = NF sizes
% column 8 = # of NFs

% Total Cost comparison
% use SRL3 inventory cost for all solutions to get an
% apples to apples comparison
TC_cost_data=Output.Total_cost2;

% --- Inventory cost ----
Inv_cost_data=[];
for i=1:3
Inv_cost_data(:,i)=Output.Inv_cost(:,i);
end

% --- #NF  ----
for i=1:3
NFnum_data(:,i)=Output.NFnum(:,i);
end
% --- Solution Time ----
Time_data=[];
for i=1:3
Time_data(:,i)=Output.time(:,i);
end

% ---  comparing NF sizes ----
% Output_NFsizes(data_run,Input.SRL,i)=NFsizes(i);

% SRL3
NF_sizes=[];
for i=1:data_run
    for j=1:length(Output_NFsizes(i,1,:))
        if Output_NFsizes(i,1,j)==0
        else
        NF_sizes(i,j)=Output_NFsizes(i,1,j);
        end
    end
end
NF_sizes_SRL3=sort(NF_sizes,2,'descend');
NF_sizes_SRL3_avg=mean(NF_sizes_SRL3);
% SRL2
NF_sizes=[];
for i=1:data_run
    for j=1:length(Output_NFsizes(i,2,:))
        if Output_NFsizes(i,2,j)==0
        else
        NF_sizes(i,j)=Output_NFsizes(i,2,j);
        end
    end
```

```
end
NF_sizes_SRL2=sort(NF_sizes,2,'descend');
NF_sizes_SRL2_avg=mean(NF_sizes_SRL2);
% SRL1
NF_sizes=[];
for i=1:data_run
    for j=1:length(Output_NFsizes(i,3,:))
        if Output_NFsizes(i,3,j)==0
        else
        NF_sizes(i,j)=Output_NFsizes(i,3,j);
        end
    end
end
NF_sizes_SRL1=sort(NF_sizes,2,'descend');
NF_sizes_SRL1_avg=mean(NF_sizes_SRL1);


% -----  Comparing NF locations  -----
NF_same=zeros(1,data_run);
NF_same_NoRho=zeros(1,data_run);
NF_near=zeros(1,data_run);
NF_same_percent_SRL1=zeros(1,data_run);
NF_same_percent_SRL2=zeros(1,data_run);
NF_near_percent_SRL1=zeros(1,data_run);
NF_near_percent_SRL2=zeros(1,data_run);
NF_close=30;

for i=1:data_run
    %squeeze locations from ND array
    NFloc=Output_NFloc(i,1,:,:);
    NFloc_SRL3=squeeze(NFloc); % SRL3

    NFloc=Output_NFloc(i,2,:,:);
    NFloc_SRL2=squeeze(NFloc); % SRL2

    NFloc=Output_NFloc(i,3,:,:);
    NFloc_SRL1=squeeze(NFloc); % SRL1

    %compares using exact/binary (0-1) comparison
    % SRL1 vs. SRL3
    for j=1:length(NFloc_SRL1)
        for k=1:length(NFloc_SRL3)
            if NFloc_SRL1(j,1)==0
            else
            NF_equal=isequal(NFloc_SRL1(j,:),NFloc_SRL3(k,:));
            NF_same(i)=NF_same(i)+NF_equal;
            end
        end
    end
    NF_same_percent_SRL1(i)=NF_same(i)/Output.NFnum(i,3)*100;
```

```
    %compare using binary (0-1) comparison
    % SRL2 vs. SRL3
    NF_same(i)=[0];
    if length(NFloc_SRL2)<=length(NFloc_SRL3)
        for j=1:length(NFloc_SRL2)
            for k=1:length(NFloc_SRL3)
                if NFloc_SRL2(j,1)==0
                else
                NF_equal=isequal(NFloc_SRL2(j,:),NFloc_SRL3(k,:));
                NF_same(i)=NF_same(i)+NF_equal;
                end
            end
        end
        NF_same_percent_SRL2(i)=NF_same(i)/Output.NFnum(i,2)*100;
    else
        for j=1:length(NFloc_SRL3)
            for k=1:length(NFloc_SRL2)
                if NFloc_SRL3(j,1)==0
                else
                NF_equal=isequal(NFloc_SRL3(j,:),NFloc_SRL2(k,:));
                NF_same(i)=NF_same(i)+NF_equal;
                end
            end
        end
        NF_same_percent_SRL2(i)=NF_same(i)/Output.NFnum(i,1)*100;
    end

%  NF comparison - compare using distance
    %compare using binary (0-1) comparison
    % SRL1 vs. SRL3
    NF_near(i)=[0];
    for j=1:length(NFloc_SRL1)
        f_one=0;
        for k=1:length(NFloc_SRL3)
            if NFloc_SRL1(j,1)==0
            else
                 if f_one==0;
                     if min(dists(NFloc_SRL1(j,:),NFloc_SRL3(k,:),'mi'))<=
NF_close;
                        NF_near(i)=NF_near(i)+1;
                        f_one=1;
                     end
                 end
            end
        end
    end
    NF_near_percent_SRL1(i)=NF_near(i)/Output.NFnum(i,3)*100;
    %compare using binary (0-1) comparison
    % SRL2 vs. SRL3
    NF_near(i)=[0];
```

```
    if length(NFloc_SRL2)<=length(NFloc_SRL3)
        for j=1:length(NFloc_SRL2)
            f_one=0;
            for k=1:length(NFloc_SRL3)
                if NFloc_SRL2(j,1)==0
                else
                    if f_one==0;
                                if
min(dists(NFloc_SRL2(j,:),NFloc_SRL3(k,:),'mi'))<= NF_close;
                                   NF_near(i)=NF_near(i)+1;
                                   f_one=1;
                                end
                    end
                end
            end
        end
        NF_near_percent_SRL2(i)=NF_near(i)/Output.NFnum(i,2)*100;
    else
        for j=1:length(NFloc_SRL3)
            f_one=0;
            for k=1:length(NFloc_SRL2)
                if NFloc_SRL3(j,1)==0
                else
                    if f_one==0;
                          if
min(dists(NFloc_SRL3(j,:),NFloc_SRL2(k,:),'mi'))<= NF_close;
                             NF_near(i)=NF_near(i)+1;
                             f_one=1;
                          end
                    end
                end
            end
        end
        NF_near_percent_SRL2(i)=NF_near(i)/Output.NFnum(i,1)*100;
    end
end
NF_same_percent_SRL1= NF_same_percent_SRL1';
NF_same_percent_SRL2= NF_same_percent_SRL2';
NF_near_percent_SRL1= NF_near_percent_SRL1';
NF_near_percent_SRL2= NF_near_percent_SRL2';

% capture the 4 percentages in a single matrix
NF_compare=[NF_same_percent_SRL2 NF_same_percent_SRL1 NF_near_percent_SRL2
NF_near_percent_SRL1];
```

Data generation procedure:
File name: Paper2_east 2

```
% Data Generation for Paper 2 - SS comparison
% Provides 5 digit zip data for east coast problems

% Created by Michael Bucci, Feb. 2008

function [Input] = Paper2_east2(Input)

% These two lines of code create the raw 5 digit zip code data
%[Input.retloc, Input.retwt] = uszip5('XY','Pop',uszip5('isCUS') &
uszip5('Pop') > 100 & sub(uszip5('XY'),'#(:,1)')>-90&
sub(uszip5('XY'),'#(:,1)')<-80);
%save ('5digitzip_Paper2_east2','Input');

% This line of code for east coast to 90 longitude for 5 digit zip code
data
%[Input.retloc, Input.retwt] = uszip5('XY','Pop',uszip5('isCUS') &
uszip5('Pop') > 100 & sub(uszip5('XY'),'#(:,1)')>-90);


load ('5digitzip_Paper2_east2')
val=100;  % # of customers in smaller problem
%Input.FC=75000;  % fixed facility cost;
Input.FC=1; % to avoid adding facilities

Input.L=6; %constant replenishment lead time (weeks)

% WEIGHTED RANDOM PERMUTATION
   cust_val=wtrandperm(Input.retwt,val);
   XY_test=Input.retloc(cust_val,:);  % customer set
   retwt_test=Input.retwt(cust_val); % reduced customer location set

% RANDOM PERMUTATION
%      cust_val=randperm(length(Input.retwt));
%      XY_test=Input.retloc(cust_val(1:100),:);  % customer set
%      retwt_test=Input.retwt(cust_val(1:100)); % reduced customer
location set

totalpop=sum(retwt_test);
retwt_test=retwt_test/totalpop;

Input.retloc=XY_test;  % update Input

% WEIGHTED DEMAND
%    Input.retwt=retwt_test;
%    Input.XY=Input.retloc;
```

```
% EQUAL DEMANDS
    Input.retwt=ones(val,1).*(1/val);  % all equal customer values
    Input.XY=Input.retloc;

% product cost and transportation data
Input.TLyear=100;
Input.prodvol=Input.TLyear*100;
volume=Input.prodvol;
Input.p='mi';  % dists parameter
% TL cost per mile
    %Input.TLcostpermile=2;
    Input.TLcost=2;
Input.avgTLmiles=245;  % based on analysis for network this was average
% truckload transportation distance traveled

% secuity factor
Input.circuity=1.2;
Input.XYf=Input.XY; % possible facility locations = all customer locations
Input.XYz=Input.XY; % for ALA problem
XYall=Input.XY; % for testing

% ----- SAMPLE DATA SET  -----------------------
%sample facility locations to test model
%Input.prodvol=sum(Input.proddata(:,6));
XYtest=[];
retwttest=[];
volume=[0];
%val=60;
volume=Input.prodvol;

% ----------  random selection of smaller subset  ----------------;
%Input.XYf=XYtest;

distance=dists(Input.XY,Input.XY,'mi');
Input.avgTLmiles= (mean(mean(distance)));

% ------- end of sample routine -------------
distance=dists(Input.XY,Input.XY,'mi');
C=distance*Input.TLcost*Input.circuity*Input.TLyear;  % cost matrix
for i=1:size(Input.XY,1)
      Input.C(:,i)=C(:,i)*Input.retwt(i,1);  % add retail wt to n x m
      %variable cost matrix,
      % where C(i,j) is the cost of serving EF j from NF i
      %includes both retail weight and distance
end
k=ones(1,length(Input.XYf))*Input.FC; %fixed facility charge are all equal
Input.k=k;
Input.retvol=Input.retwt*Input.prodvol; %yearly volume
Input.retvol_week=Input.retvol/52;
```

```
% ------------ I/T Ratios -----------
% STEP 1: First calculate the transportation cost for single facility
% solution

[TC,Input.y] = min(sum(Input.C,2));% Determine first NF location
TC_trans=TC/3;

IT_ratio=0.3; % set inventory cost / transportation cost
TC_inv=TC_trans*IT_ratio;

% ------ Inventory Parameters  ---------------------
% demand variances
% initial assumption is the coefficient of variation for all customers
% is the same -  use this to get
% variances for each demand point
Input.h=0.30;  % holding cost per unit per year
Input.CSL=1.645; % 95% CSL for normal distribution

Input.cv=zeros(length(Input.retvol_week),1)';
%For fixed c.v. value
%     cv=0.2;  % coefficent of variation (std.dev/mean)
%     Input.cv(:)=cv;
%     for i=1:length(Input.retvol_week)
%         Input.retvar(i)=(cv*Input.retvol_week(i))^2;
%     end

% For random c.v. values
    for i=1:length(Input.retvol_week)
        a=0.2;
        b=0.35;
        cv_rand= a + (b-a).*rand(1,1);
        % for random cv
        Input.cv(i)= cv_rand;
        Input.retvar(i)=(cv_rand*Input.retvol_week(i))^2;
    end
% %
% --- correlation coefficient ---
% need correlation coefficient for each demand pair
Input.rho=zeros(length(Input.retwt),length(Input.retwt));

% create covariance (cov) matrix
Input.cov=zeros(length(Input.retwt),length(Input.retwt));
for i=1:length(Input.retwt)-1
    for j=i+1:length(Input.retwt)
        %Input.cov(i,j)= (Input.retvar(i)^0.5)*(Input.retvar(j)^0.5);
        Input.cov(i,j)=
Input.rho(i,j)*(Input.retvar(i)^0.5)*(Input.retvar(j)^0.5);
        Input.cov(j,i)=Input.cov(i,j);
    end
end
```

```
% determine unit cost to meet I/T ratio
    % add inventory cost get TC for single facility solution
    % determine DC standard deviation over the lead time for single DC
location

    %based on a estimated average value of c.v.;
    %week_vol=sum(Input.retvol_week);

    DC_step= Input.retvol_week'.*0.2;
    DC_ss=(Input.CSL)*((Input.L)^0.5)*((sum(DC_step.*DC_step))^0.5);

    Input.prodcost=(TC_inv)/(DC_ss*Input.h);

    Input.unitcost=Input.prodcost;
    Input.prodcostyr=Input.prodvol*Input.prodcost;

%  ------------------
% transportation cost for year
TCavg_sample = Input.TLcost*Input.avgTLmiles*Input.TLyear;
% product cost per year
PCavg_sample = Input.prodvol*Input.unitcost;
```

Construction subset generation - constructive ADD procedure for generation of a subset of possible facility locations to be used in the ADD procedure within the meta-heuristic

File name: UFCLLadd_ss

```
function [Input] = UFCFLadd_ss(Input)
%UFLADD Add construction procedure for uncapacitated facility location.
% Developed by Michael Kay.  Modified by Mike Bucci April 2006

   % Sample Data - see Daskin textbook
        % Programming of M.S. Daskin Example
        % Network and Discrete Locations, 1995
        % Chapter 7, section 7.2 - Figure 7.4
        % Add Heuristic for facility locations in Network
        % Design problem with fixed facility charges.

% This algorithm uses Daskin's add procedure to determine a subset of
% facilities out of the total number of facilities to use in the Whitaker
% procedures

% Input Error Checking
**************************************************
% error(nargchk(2,3,nargin));

[n,m] = size(Input.C);
Input.k = Input.k(:);

if nargin < 3, Input.y = []; end
%
*********************************************

if isempty(Input.y)
   [TC,Input.y] = min(sum(Input.C,2) + Input.k);      % Determine first NF
location
        % min (sum rows and add cost of adding a facility)
else
   TC = sum(Input.k(Input.y)) + sum(min(Input.C(Input.y,:),[],1)); % if
existing locations are provided
    % sum the minimum in each row of y (finds closest facility) and add
the sum of k(y) which
    % is the total NF cost
end
%Input.la_cycles=Input.la_cycles+1;  % counting allocation cycles
ny = 1:n;  % n= # of potential NF locations
ny(Input.y) = [];

done = 0;
TC=Inf;
```

```
% stopping criteria for subset
NFnum=round(length(Input.XYf)*Input.percent);

while ~done
   TC1 = Inf;  % set initial TC to infinity

   for i = 1:length(ny)
      yi = [Input.y ny(i)]; % adds cost of NF to
      TCi = sum(Input.k(yi)) + sum(min(Input.C(yi,:),[],1));% using
current NF locations
            % sum the minimum in each row of y (finds closest facility)
and add the sum of k(y) which
            % is the total NF cost to get a total cost for current state
      if TCi < TC1 % if better solution found
         i1 = i;
         TC1 = TCi;
      end
   end

   % add the next best facility to solution
   Input.y = [Input.y ny(i1)]; % set the NF locations to
   ny(i1) = [];
   TC = TC1;

   % check to see if we stop adding facilities
   if length(Input.y)==NFnum
        done = 1;  % stop loop if total cost is not lower
   end
end

Input.y = sort(Input.y);  % final locations of NF
X = logical(sparse(Input.y(argmin(Input.C(Input.y,:),1)),1:m,1,n,m));
%final allocation
```

Meta-heuristic that combines constructive ADD procedure with ALA procedure

File name: Paper2_heuristic_a

```
function[Input,Whitaker2] = Paper2_heuristic_a(Input, Whitaker2);

% Hybrid construction procedure for uncapacitated facility location with
    % correlation
    % based  on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, March 2008

% STEP 0: Find single facility the minimizes total distribution costs of
% the network (transportation, and Safety stock (SS) costs)

% STEP 1: To add a facility... For each candidate median (each DC site not
in
% current solution) place a NF at the median location and
% allocate demand based on transportation cost only. Then add inventory
costs
% Calculate total cost of this solution.
% Choose the solution that has lowest total cost.

% STEP 2: reallocate customers based on all costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement

% STEP 3: relocate DCs based on allocation

% Repeat step 2 and 3 until no improvement
% Return to step 1 and continue iteration until no improvement

% ------------------------------------------------------
% STEP 0
time1=clock;
[Input,TC_one_DC,X] = Paper2_initialize(Input);
%TC_one_DC % for testing
time2=clock;
Input.time = Input.time + etime(time2, time1);

% Keep data on best solution for each NF level (# of facilities)
nNF=1;
Input.TCall_Whitaker(1)=TC_one_DC;
Input.TCall_Whitaker;
Whitaker2.TC_data(nNF) = TC_one_DC; % add initial cost to cost data
Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
Whitaker2.NF_num(nNF)=length(Input.y);
Whitaker2.X_data{nNF,1} = X; % allocation data
Whitaker2.y_data{nNF,1} = Input.y; % y values
X_input=X;
```

```
%Whitaker2.trans_cost_data(nNF)=sum(sum(Input.C.*X));
[inv_cost]= W2_SS_srl3(Input,X_input);
inv_ss=(inv_cost)/(Input.prodcost*Input.h);
Whitaker2.IC_SRL3(nNF)=inv_ss;

[inv_cost]= W2_SS_srl2(Input,X_input);
inv_ss=(inv_cost)/(Input.prodcost*Input.h);
Whitaker2.IC_SRL2(nNF)=inv_ss;

[inv_cost]= W2_SS_srl1(Input,X_input);
inv_ss=(inv_cost)/(Input.prodcost*Input.h);
Whitaker2.IC_SRL1(nNF)=inv_ss;

nNF=nNF+1;
Xa=X;
Xin=X;
% -----------------------------------------------------
% Loop for Steps 1,2,3
extend=0;
done5 = 0;
loopcount=1;
while ~done5
    % -----------------------------------------------------
    % STEP 1
    % To add a facility
    time1=clock;
    [Input,X,TC,W] = Paper2_add(Input);
    %TC % for testing
    time2=clock;
    Input.time = Input.time + etime(time2, time1);
    Input.TCall_Whitaker(end+1)=TC;
    Input.TCall_Whitaker;
    % for TABU search
    Input.y_last = Input.y;
    length(Input.y) % show for testing
    Input.W_last=W;
    Input.X_last=X;

        % plot solution
        Wcurrent=W(Input.y,:);
        %alaplot(Input.XYf(Input.y,:),Wcurrent,Input.XY,'W2 - Add');
        if length(Input.y)==2;
            Input.y_last = Input.y;  % for TABU search
            Input.W_last=W;
            Input.X_last=X;
        end
        TC_NF=TC; % best solution with next NF added
        TCin=TC;
        TCin_add=TCin; % show for testing
        Xin=X;
    % -------------------------------------------------
```

```
    % Loops for STEP 2,3
    % TC = best solution so far
    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
        loops=1;
        while ~done3 % loop until no improvement in the allocation
            % STEP 2: reallocate customers
            %TCin=TC;
            %[Input,W,X,TC] = W2_allocate_all_unitcost_rho_2(Input,X,TC);
            %randomly sequence the relocation attempts
            [Input,W,X,TC] = Paper2_allocate(Input,X,TC); % all,
            % check for improvement
            %TC % show for testing
            if TCin > TC
                TCin = TC;
                Xin=X;
                loops=2;
                allocate_improve=1; % for testing
                Wcurrent=W(Input.y,:);
                sum(Wcurrent,2); % show for testing
               %alaplot(Input.XYf(Input.y,:),Wcurrent,Input.XY,'Whitaker2
-
               %Allocate Loop');
            else
                done3 = 1;
            end
            Input.TCall_Whitaker(end+1)=TC;
            Input.y;
        end % allocation loop
        TCin_alloc=TCin; % show for testing

        % STEP 3: relocate DCs based on allocation
        done_loc=0;
        TC=TCin;
        while ~done_loc
            TCin=TC;
            [Input,X,W,TC] = Paper2_locate(Input,X,W,TC);
            if TCin>TC  % change in TC in location solution
                loops=2;
                TCin=TC;
                Xin=X;
            else
                done_loc=1;
            end
            Input.TCall_Whitaker(end+1)=TCin;
        end
        TCin_loc=TCin; % show for testing
        %Input.TCall_Whitaker;
        % For tabu search
```

```
        Input.y_last = Input.y;
        Input.W_last=W;
        Input.X_last=X;

        if loops==1
            done4=1;
        end
        Wcurrent=W(Input.y,:);
        sum(Wcurrent,2); % show for testing

      end % done4 loop - Repeat step 2 (allocation) and step 3 (location)
until no improvement

    %alaplot(Input.XYf(Input.y,:),Wcurrent,Input.XY,'W2 - DROP unused
NF');
    TC=TCin; %-sum(Input.k(remove_i));
    %TC % show for testing
    Input.TCall_Whitaker(end)=TC;

    % ------------------------------------------------
    % Compare best result from Step 1,2,3 iteration with p+1 NFs with best
    % result from previous run with p NFs
    %if Whitaker_TC_data(nNF) >= min(Whitaker_TC_data)
    if TC >= min(Whitaker2.TC_data)
        %TC
        %min(Whitaker2.TC_data)
        done5=1; % stop when no improvement
        extend=extend+1;
            if extend<4 % go past the best solution by a few NF to avoid
                 % a premature exit of the program and to have additional
                 % potential solutions to explore
             done5=0;
            end
    end

    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori
    Whitaker2.TC_data(nNF) = TC; % Total cost data
    Input.y;
    Whitaker2.NFloc_data{nNF,1} = Input.XYf(Input.y,:); % NF locations
    Whitaker2.y_data{nNF,1} = Input.y; % y values
    Whitaker2.X_data{nNF,1} = X; % allocation data
    Whitaker2.NF_num(nNF)=length(Input.y);

    % Get Transportation and Inventory costs
    X_input=X;
    [inv_cost]= W2_SS_srl3(Input,X_input);
    Whitaker2.inv_cost_data(nNF)=inv_cost;
    %calculate transportation cost
    Whitaker2.trans_cost_data(nNF)=sum(sum(Input.C.*X_input));
```

```
    [inv_cost]= W2_SS_srl3(Input,X_input);
    fixed_cost_addrho=sum(Input.k(Input.y));

Whitaker2.TC_withrho(nNF)=Whitaker2.trans_cost_data(nNF)+inv_cost+fixed_co
st_addrho;
    %nNF
    [inv_cost]= W2_SS_srl3(Input,X_input);
    fixed_cost_addrho=sum(Input.k(Input.y));

    % collect data on inv. cost for each NF level
        inv_ss=(inv_cost)/(Input.prodcost*Input.h);
        Whitaker2.IC_SRL3(nNF)=inv_ss;  %SRL3

        [inv_cost]= W2_SS_srl2(Input,X_input);
        inv_ss=(inv_cost)/(Input.prodcost*Input.h);
        Whitaker2.IC_SRL2(nNF)=inv_ss;  % SRL2

        [inv_cost]= W2_SS_srl1(Input,X_input);
        inv_ss=(inv_cost)/(Input.prodcost*Input.h);
        Whitaker2.IC_SRL1(nNF)=inv_ss;  %SRL1

    Wcurrent=W(Input.y,:);
    NFsizes=sum(Wcurrent,2);
    Whitaker2.IC_SRL3;
    xyz=NFsizes';
    Whitaker2.NFsizses{nNF,1} =xyz;
    nNF=nNF+1;
    if nNF-1==length(Input.XYf_2)
        done5=1; % all possible NFs locations have a NF
    end
    if nNF-1==20
        done5=1; % all possible NFs locations have a NF
    end

end % end for loop for steps 1, 2, and 3
```

Initialize model - code to find single facility solution

File name: Paper2_initialize

```
function[Input,TC_one_DC,X] = Paper2_initialize(Input);

% Code by Mike Bucci, November 2007
% based on Whitaker (1985) ADD/DROP formulation.

% STEP 0: Find single facility that minimizes total distribution costs of
% the network (transportation + inventory costs) using
% correlation coefficient

Input.y=[];
[n,m] = size(Input.C);
Input.k = Input.k(:);

[TC,Input.y] = min(sum(Input.C,2) + Input.k);% Determine first NF location
TC_one_DC=TC;

Input.la_cycles=Input.la_cycles+1; %allocation count

% update C matrix to include both transportation and production scale
costs
%Ceos=Input.C;

[TC_one_DC,Input.y] = min(sum(Input.C,2) + Input.k);% Determine first NF
location

% add inventory cost get TC for single facility solution
% determine DC standard deviation over the lead time for single DC
location
% get facility sizes and SS costs
X_input=zeros(length(Input.retwt));
X_input(Input.y,:)=1;

% establish SS1 based on
% explicit calculation of single facility
% solution with correlation

%     [inv_cost]= W2_SS_srl3(Input,X_input);
%     DC_ss=inv_cost/(Input.prodcost*Input.h);
%     Input.SS1=DC_ss;


    % if we want to exclude cov. in the SS1 calculation
    save_cov=Input.cov;
    Input.cov=zeros(length(Input.retwt),length(Input.retwt));
```

```
    [inv_cost]= W2_SS_srl3(Input,X_input);
    DC_ss=inv_cost/(Input.prodcost*Input.h);
    Input.SS1=DC_ss;
    Input.cov=save_cov;

if Input.SRL==1
    [inv_cost]= W2_SS_srl3(Input,X_input);
end
if Input.SRL==2
    [inv_cost]= W2_SS_srl2(Input,X_input);
end
if Input.SRL==3
    [inv_cost]= W2_SS_srl1(Input,X_input);
end

% ss cost= ss*unit cost*holding cost factor
%DC_ss=inv_cost/(Input.prodcost*Input.h);
%Input.SS1=DC_ss;
TC_one_DC=TC_one_DC+inv_cost;
X=X_input;


% Outputs
Input;
TC_one_DC;
X;
```

ADD construction procedure – used to add the next facility to the solution

File name: Paper2_add

```
function[Input,X,TC,W] = Paper2_add(Input);

% Hybrid construction procedure for uncapacitated facility location with
    % eos based on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, November 2006 - UPDATED Oct 2007 to include rho,
%   UPDATED March 2008 to use SRL for inventory calculation

% STEP 1

[n,m] = size(Input.C);
TC1=Inf;
TC=TC1;
% current cost

ny=setdiff(Input.XYf_2,Input.y);  % ny= potential NF locations that are in
the subset, but not in the
% the current solution.
NFadd=[];
for i = 1:length(ny)  % for loop to add each median facility in the subset
and choose
    % the best addition
    yi = [Input.y ny(i)];
    Input.y=yi;
    % Allocate based on distance only
    X = logical(sparse(yi(argmin(Input.C(yi,:),1)),1:m,1,n,m));
    % now add inventory costs to the solution
    X=full(X);
    X_input=X;

    if Input.SRL==1
        [inv_cost]= W2_SS_srl3(Input,X_input);
    end
    if Input.SRL==2
        [inv_cost]= W2_SS_srl2(Input,X_input);
    end
    if Input.SRL==3
        [inv_cost]= W2_SS_srl1(Input,X_input);
    end
    %calculate transportation cost
    trans_cost=sum(sum(Input.C.*X_input));
    % calculate fixed facility cost
    fixed_cost=sum(Input.k(yi));

    % total cost
     TCi=trans_cost+inv_cost+fixed_cost;
```

```
    if TCi < TC1 % if better solution found
     NFadd = ny(i);
     TC1 = TCi;
     Xbest=X;
    end
    Input.y(end)=[];

%Input.construct_cycles_2=Input.construct_cycles_2+(length(Input.y)+1)*siz
e(Input.C,2);
    Input.construct_cycles_2=Input.construct_cycles_2+ size(Input.C,2);
    %Input.construct_cycles_2=Input.construct_cycles_2+1;

end % for loop to try a different facility
% ----------
% add the NF facility to the current best solution
Input.y = [Input.y NFadd];
TC = TC1;
X=Xbest;
%X_input=X;
%[inv_cost]= W2_SS_rho(Input,X);
for wi =1:size(X,1)
    for wj=1:length(Input.retwt)
        W(wi,wj)=Input.retwt(wj)*X(wi,wj); % create new W matrix for best
solution
    end
end
% outputs
 X;
 TC;
 W;
```

Allocation procedure – procedure that allocates customers to facility locations

File name: Paper2_allocate

```
function [Input,W,X,TC] = Paper2_allocate(Input,X,TC);

% Code by Mike Bucci, November 2006, updated October November 2007 to
% include correlation coefficient and SS comparison
% based on Whitaker (1985) ADD/DROP formulation - STEP 2

% STEP 2: reallocate customers based on transportation, fixed, and scale
costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement%   performs the allocation of
demand to the DCs based
%   on transportation cost and an economy of scale production charge

% NFloc = NF locations (does not change in this function)
% W1= allocation weight matrix (0 if NF does not serve EF, otherwise
%   the EF weight
% TC1= Total cost after new allocation
% eosTC= total eos cost (for location subroutine)

%Input.alloc_cycles=Input.alloc_cycles+1; %allocation count

% Allocation subroutine for W2 algorithm with correlation coefficient
% only consider facilities on the convex hull for re-allocation

% --- 2. Try to reallocate all customers ----------
NFloc=Input.XYf(Input.y,:);
n=size(NFloc,1);

X_input=X;
TC_input=TC;
TCa=TC;
% cust_list=randperm(length(Input.XYf)); % random allocation of customers

for i=1:length(Input.XYf) % all customers in allocation
    for j=1:n  % n= # of facilities
        % get current facility for the customer
        NF_input=[0];
        for b=1:n
            if X_input(Input.y(b),i)==1
               NF_input=Input.y(b);
            end
        end
        if Input.y(j)~=NF_input % do not check current location again
            % move 1 facility at a time to other current facilities
            % keep solution if cost is lowered
```

```
% *** NOTE: compares total cost for entire solution ****

% move customer in X matrix
X_test=X_input;
X_test(:,i)=0; % no retail allocation
X_test(Input.y(j),i)=1; % allocate to another facility

%count allocation cycles;
Input.alloc_cycles=Input.alloc_cycles+1;
% ---- Total cost for "input" location ---------
% calculate inventory cost
if Input.SRL==1
    [inv_cost]= W2_SS_srl3(Input,X_input);
end
if Input.SRL==2
    [inv_cost]= W2_SS_srl2(Input,X_input);
end
if Input.SRL==3
    [inv_cost]= W2_SS_srl1(Input,X_input);
end
%calculate transportation cost
trans_cost=sum(sum(Input.C.*X_input));
% calculate fixed facility cost
fixed_cost=sum(Input.k(Input.y));

% total cost
TC_input=trans_cost+inv_cost+fixed_cost;

% ---- Total cost for "test" location ---------
% calculate inventory cost
if Input.SRL==1
    [inv_cost]= W2_SS_srl3(Input,X_test);
end
if Input.SRL==2
    [inv_cost]= W2_SS_srl2(Input,X_test);
end
if Input.SRL==3
    [inv_cost]= W2_SS_srl1(Input,X_test);
end
%calculate transportation cost
trans_cost=sum(sum(Input.C.*X_test));
% calculate fixed facility cost
fixed_cost=sum(Input.k(Input.y));

% total cost
TC_test=trans_cost+inv_cost+fixed_cost;

% ------ total cost comparison  --------------
TC_input;
TC_test;
```

```
            %Input.y
            % if TC lowered, update allocation
            if TC_test<TC_input
                X_input=X_test;
                %X_input(NF_input,i)=0;
                %X_input(Input.y(j),i)=1;
                Input.alloc_change=Input.alloc_change+1;
            end
        end % end for if loop
    end % go to next facility
end % go to next customer

% ------------- OUTPUT DATA ------------------
% new X matrix
Xa=X_input;
X=X_input;
% new weight matrix (Wa)
Wa=zeros(size(Input.XYf,1),size(Input.XYf,1));
for p=1:length(Input.retloc)
    for q=1:n
        if Xa(Input.y(q),p)==1
            Wa(Input.y(q),p)=Input.retwt(p); % weight matrix
        end
    end
end
% new total cost(Ta)
%calculate inventory cost
if Input.SRL==1
    [inv_cost]= W2_SS_srl3(Input,X_input);
end
if Input.SRL==2
    [inv_cost]= W2_SS_srl2(Input,X_input);
end
if Input.SRL==3
    [inv_cost]= W2_SS_srl1(Input,X_input);
end

%calculate transportation cost
trans_cost=sum(sum(Input.C.*X_input));
% calculate fixed facility cost
fixed_cost=sum(Input.k(Input.y));

% total cost
TCa=trans_cost+inv_cost+fixed_cost;

% NFloc
NFalloc=Input.XYf(Input.y,:);
Walloc=Wa(Input.y,:);
TCalloc=TCa;  % used to check progression of TC
%Wa=W1;
%alaplot(NFloc,Walloc,Input.retloc,'Whitaker Step 2 - Allocate');
```

```
%pplot(NFalloc,'bo')
X=Xa;
W=Wa;
TC=TCa;
```

Location Procedure

File name: Paper2_locate

```
% Mike Bucci Dec. 2007

function [Input,X,W,TC]= Paper2_locate(Input,X,W,TC);

% based on Whitaker (1985) ADD/DROP formulation - STEP 3
% STEP 3: relocate DCs based on allocation - possible candidates for NF
are
% all facilities currently within the subgraph(within the allocation)

NFloc=Input.XYf(Input.y,:);

% Get neighbors
TRI3=Input.TRI2;
count=0;
for i=1:length(Input.TRI2)
    ok=0;
    for j=1:length(Input.y)
        if Input.TRI2(i,1)== Input.y(j)
            ok=1;
            j=length(Input.y);
        end
    end
    if ok==1
    else
        TRI3(i-count,:)=[];
        count=count+1;
    end
end

% delete any NF that are in the current solution from the neighbor list
count=0;
length3=length(TRI3);
TRI3b=TRI3;
for i=1:length3
    ok=0;
    for j=1:length(Input.y)
        if TRI3b(i,2)== Input.y(j)
            ok=1;
            j=length(Input.y);
        end
    end
    if ok==1
        TRI3(i-count,:)=[];
        count=count+1;
    end
end
```

```
% Local Neighborhood search of all NF locations in the k=1 neighborhood
ichange=0;
for i=1:length(Input.y)
    loc_test=0; %tabu variable

    % did allocation change for the NF?  If so, check location of all EF
in neighborhood
    change=1;
    loc_change= isequal(W(Input.y(i),:),Input.W_last(Input.y(i),:));
    if loc_change==1
        loc_test=1;  % allocation changed
    end

    % get transportation cost + fixed costs for current NF location
    trans_cost=sum(Input.C(Input.y(i),:).*X(Input.y(i),:));
    % calculate fixed facility cost
    fixed_cost=sum(Input.k(Input.y(i)));

    % total cost
    TC_CF=trans_cost+fixed_cost;

    % try to move NF to all possible NF in the neighborhood
    better_y=0;
    for j=1:length(TRI3)
        loc_test2=0; % tabu variable
        if Input.y(i)==TRI3(j,1)
            if Input.y(i)~=Input.y_last(i) %facility location has changed
                loc_test2=1;
                Input.y_last(i)=Input.y(i); % update Input.y
            end

            % get transportation cost + fixed costs for test NF location
            trans_cost=sum(Input.C(TRI3(j,2),:).*X(Input.y(i),:));

            % calculate fixed facility cost
            fixed_cost=sum(Input.k(TRI3(j,2)));

            % total cost
            TC_TF=trans_cost+fixed_cost;

                if TC_CF>TC_TF
                    better_y=j;
                end
        end
    end
    if better_y~=0; %shift all data to new location
        W(TRI3(better_y,2),:)=W(Input.y(i),:);     % shift weighted
allocation to the new location
        Input.W_last(TRI3(better_y,2),:)=W(Input.y(i),:);
```

```
        W(Input.y(i),:)=0;          % weighted allocation at "old"
facility is now zero
        Input.W_last(Input.y(i),:)=0;
        X(TRI3(better_y,2),:)=X(Input.y(i),:);     % shift (0,1)allocation
to the new location
        X(Input.y(i),:)=0;          % allocation at "old" facility is now
zero
        Input.y(i)=TRI3(better_y,2);               % move to the new
location
        ichange=1;
        Input.loc_change=Input.loc_change+1;
    end
end

% update total cost if locations have changed
if ichange==1;
    ichange;
    % new weight matrix (Wa)
    %calculate inventory cost
    X_input=X;
    %inv_cost=0;
    if Input.SRL==1
        [inv_cost]= W2_SS_srl3(Input,X_input);
    end
    if Input.SRL==2
        [inv_cost]= W2_SS_srl2(Input,X_input);
    end
    if Input.SRL==3
        [inv_cost]= W2_SS_srl1(Input,X_input);
    end

    %calculate transportation cost
    trans_cost=sum(sum(Input.C.*X_input));

    % calculate fixed facility cost
    fixed_cost=sum(Input.k(Input.y));

    % total cost
    TC=trans_cost+inv_cost+fixed_cost;

    % NFloc
    NFloc=Input.XYf(Input.y,:);
    Walloc=W(Input.y,:);
    TCloc=TC;  % used to check progression of TC
    %Wa=W1;
    %alaplot(NFloc,Walloc,Input.retloc,'Whitaker Step 2 - Allocate');
end

NFlocation=NFloc;
Wloc=sum(sum(W));
```

```
TCloc=TC;  % used to check progression of TC
%  -----------   output   --------------------
%NF final locations
NFloc=Input.XYf(Input.y,:);
% Normalize lon-lat pairs
NFloc = normlonlat(NFloc);
X;
W;
TC;
Wcurrent=W(Input.y,:);
```

"Square Root Law" calculation – SRL_1

File name: W2_SS_srl1

```
% Mike Bucci March 2008

function [inv_cost]= W2_SS_srl1(Input,X_input);

% Safety stock inventory calculation using the Square Root Law
%   Input.y provides the number of facilities in the solution

DC_ss = Input.SS1*(length(Input.y)^0.5);% SRL

DC_ss_total_cost=DC_ss*Input.prodcost*Input.h;
inv_cost=DC_ss_total_cost; % inventory cost for solution
```

General Concave Cost function calculation – SRL_2

File name: W2_SS_srl2

```
% Mike Bucci March 2008

function [inv_cost]= W2_SS_srl2(Input,X_input);

% Safety stock inventory calculation using a concave cost function, that
is predicated on facility size, to determine
% inventory cost
% Input.y provides facilities in current solution
% X_input provides allocation

% get facility sizes and SS costs
for p=1:length(Input.y)
    DC_size=sum(X_input(Input.y(p),:).*Input.retwt')*Input.prodvol;
    DC_ss(p)= Input.SS1*(DC_size/Input.prodvol)^0.5;
end

%ss cost= ss*unit cost*holding cost factor
DC_ss_total_cost=sum(DC_ss*Input.prodcost*Input.h);
inv_cost=DC_ss_total_cost; % inventory cost for solution
```

Explicit calculation for safety stock inventory – SRL_0

File name: W2_SS_srl3

```
% Mike Bucci Dec. 2007

function [inv_cost]= W2_SS_srl3(Input,X_input);

% Safety stock inventory calculation for analysis of problems with
% inter-customer correlation of demand

%     Input provides current facility locations, retail variances,
%     and other costs/values for ss calculation
%     X_input provides allocation

%     We calculate safety stock cost based on formulation provided
%     by Chopra and Miendel , second edition, Supply Chain Management
%     text book, Chapter 11

%inv_cost=0;
DC_ss=[];
for p=1:length(Input.y)
    sum_cov=0;
    sum_var=sum((Input.retvar).*X_input(Input.y(p),:)); % sum of variances
    for q=1:length(Input.cov)-1
        for r=q+1:length(Input.cov)
            if (X_input(Input.y(p),q)==1) && (X_input(Input.y(p),r)==1)
                sum_cov=sum_cov+Input.cov(q,r);
            end
        end
    end
    sum_cov=sum_cov*2;
    % CSL x sqrt (leadtime) x sqrt(sum variance + sum covariance)
    DC_ss(p)=(Input.CSL)*(Input.L)^0.5*((sum_var+sum_cov)^0.5);
     %DC_ss(p)=(Input.CSL)*(Input.L)^0.5*((sum_var)^0.5);
        % note: do not need to multiply covariance by 2 since we
        % sum the entire matrix
end
% ss cost= ss*unit cost*holding cost factor
DC_ss_total_cost=sum(DC_ss*Input.prodcost*Input.h);
inv_cost=DC_ss_total_cost; % inventory cost for solution
```

Appendix C: Supplemental Information for Chapter 4

C.1.0 Additional data not included in journal paper

For completeness, we find optimal solutions (using CPLEX 9.0) for the problems studied with no economies of scale and compare these solutions with the heuristic solutions. For the CARE network with 300M lb the heuristic solution is 0.78% higher than the optimal solution. For the 400 customer network with 300M lb and 1800M lb the heuristic solutions are respectively 0.44% and 0.014% higher than the optimal solution. These results are comparable to the results obtained in Chapter 2 when the solution heuristic was compared to optimal solutions: the error is less than 1% in all tests, and as the number of facilities in the final solution increases the heuristics performance improves (quite significantly in this case.)

C.1.1 MATLAB code

The MATLAB functions (code) that were developed for the research in Chapter 4 are shown below. We direct the reader to the Chapter 2 appendix where functions used for Chapter 4 research are direct adaptations of functions developed in that chapter

- Top Level file
  - Data generation procedures
  - Construction subset generation – see Chapter 2 Appendix
  - Generating CPLEX text file
  - Meta-heuristic that combines constructive ADD procedure with ALA procedure
    - Initialize model – see Chapter 2 Appendix
      - Calculation of processing costs
    - ADD construction procedure – see Chapter 2 Appendix
      - Calculation of processing costs
    - Allocation procedure with VNS search improvements– see Chapter 2 Appendix
      - Calculation of processing costs
    - Location procedure with tabu search improvements – see Chapter 2 Appendix
      - Calculation of processing costs

Top Level file for evaluating the solving the carpet recycling network design problem

File name: Carpet_Model_A

```
% Facility location model for carpet recycling project
% Heuristics adapted from M. Bucci dissertation chapter 2
% Dec. 2008

% Created by Michael Bucci
clear
tic;
% ---------   generate data for analysis   ----------------------------
[Input] = Carpet_data_generation_3digitzip_A; % 3 digit zip locations
Input.percent=0.1;

%[Input] = Carpet_data_generation_CARE_non_overlap; % CARE network, 5
digit zip to determine demand weighting, but with overlap removed (by %
splitting demand equally among the CARE locations that service this zip
%Input.percent=1.0;

%[Input] = Carpet_data_generation_CARE_non_overlap_eos;
% same as above but with economies of scale data

% ----- map data -------------------
% makemap(Input.retloc)
% pplot (Input.retloc, 'k.')
%pplot (Input.retloc(i), 'bx')
%pplot (X,'go')
% ------- additional input variables  ------------------------------------
%Input.XY=Input.
%Input.XYf=Input.XY;
% restrict first facility to NC, SC, GA
    Input.fix_1se=0; % 0 = no restriction, 1=restriction
% force model to not change first facility location
    Input.fix_1=0; % 0 = no restriction, 1=restriction
% restrict model from adding too many facilities
    Input.Max_fac=12;

% -------------------------- To create text file for CPLEX ---------------
% use the below two lines of code when the scale exponent is zero.  This
% eliminates the production cost from CPLEX (reducing the variables) the
% production cost is then added later.

%  [filedone] = Carpet_textfileforCPLEX_no_eos(Input);

%-----------    Input for Hybrid and Whitaker program  -------------------
-----
% get neighbors for all possible NFs locations
TRI = delaunayn(Input.facloc); % do this upfront instead of each time
location loop is performed
```

```
Input.TRI2=tri2list(TRI);
for i=1:length(Input.TRI2) % change to positive values
    Input.TRI2(i,2)=-Input.TRI2(i,2);
end
g=length(Input.TRI2);
% add y to x option
for i=1:g
    Input.TRI2(end+1,1)=Input.TRI2(i,2);
    Input.TRI2(end,2)=Input.TRI2(i,1);
end
%%  ---- Daskin's Add used to create construction subset -------
% %tic;
Input.y=[];
%Input.time=[0];
[Input] = Carpet_add_ss(Input);
%time_Add_ss=toc;
Input.XYf_2=sort(Input.y);

% use this code to use all facilities in the subset
%   n=length(Input.facloc);
%   Input.XYf_2=1:n;

%% ---------- Run each of the SRL methods using: W2-Sw-LAT-rho  Whitaker2
with TABU in location and allocation and
% subset in construction procedure with rho for inventory calculation

%tic;
Input.y=[];
Input.alloc_cycles=[0];
Input.loc_cycles=[0];
Input.la_cycles=[0];
Input.construct_cycles=[0];
Input.construct_cycles_2=[0];
Input.time=[0];
Input.TCall_Whitaker=[];
Input.alloc_change=0;
Input.loc_change=0;
Whitaker2.TC_data=[];
Whitaker2.NFloc_data = [];
Whitaker2.y_data = [];
Whitaker2.X_data = [];
Whitaker2.degen=[0];
Whitaker2.NF_num=[];
Whitaker2.processing_cost_data=[];
Whitaker2.trans_cost_data=[];

%[Input,Whitaker2] = Carpet_LA_model(Input, Whitaker2); %without eos
[Input,Whitaker2] = Carpet_LA_model_eos(Input, Whitaker2); %with eos

for i=1:length(Whitaker2.TC_data)
    if Whitaker2.TC_data(i) == min(Whitaker2.TC_data)
```

```
        TC_Carpet = Whitaker2.TC_data(i); % lowest TC
        NF_Carpet= Whitaker2.NFloc_data{i,:}; % final NF
        X_Carpet = Whitaker2.X_data{i,:}; % final allocation
        y_Carpet = Whitaker2.y_data{i,:};
        W_Carpet =zeros(size(X_Carpet,1),size(X_Carpet,2));
        for p=1:length(Input.retloc)
            for q=1:length(y_Carpet)
                if X_Carpet(y_Carpet(q),p)==1
                    W_Carpet(y_Carpet(q),p)=Input.retwt(p); % weight
matrix
                end
            end
        end
    end

    X_Carpet_c = Whitaker2.X_data{i,:}; % final allocation
    y_Carpet_c = Whitaker2.y_data{i,:};
    Wcurrent =zeros(size(y_Carpet_c,2),size(X_Carpet_c,2));
    for p=1:length(Input.retloc)
        for q=1:length(y_Carpet_c)
            if X_Carpet_c(y_Carpet_c(q),p)==1
                Wcurrent(q,p)=Input.retwt(p); % weight matrix
            end
        end
    end
end
total_time=toc;
time_min=total_time/60;
TCall_Carpet=Whitaker2.TC_data;
% to find closest city to the factory locations
lonlat2city(Whitaker2.NFloc_data{1,:},uscity)
lonlat2city(Whitaker2.NFloc_data{2,:},uscity)
lonlat2city(Whitaker2.NFloc_data{3,:},uscity)
lonlat2city(Whitaker2.NFloc_data{4,:},uscity)
lonlat2city(Whitaker2.NFloc_data{5,:},uscity)
lonlat2city(Whitaker2.NFloc_data{6,:},uscity)

lonlat2city(Whitaker2.NFloc_data{7,:},uscity)
lonlat2city(Whitaker2.NFloc_data{8,:},uscity)
lonlat2city(Whitaker2.NFloc_data{9,:},uscity)

lonlat2city(Whitaker2.NFloc_data{10,:},uscity)
lonlat2city(Whitaker2.NFloc_data{11,:},uscity)
lonlat2city(Whitaker2.NFloc_data{12,:},uscity)

%  ----- Sensitivity analysis ----
%Calculate current total cost using demand weights from the other scenario
for i=1:length(Whitaker2.TC_data)
    Input.y=Whitaker2.y_data{i,:};
    X_test=Whitaker2.X_data{i,:}; % final allocation
    %calculate transportation cost
```

```
    trans_cost=sum(sum(Input.C.*X_test));
    % calculate fixed facility cost
    fixed_cost=sum(Input.k(Input.y));
    % Total Cost
    TC_compare(i)=trans_cost+fixed_cost;
end

X_Carpet_c = X_test; % final allocation
y_Carpet_c = Input.y;
Wcurrent =zeros(size(y_Carpet_c,2),size(X_Carpet_c,2));
for p=1:length(Input.retloc)
    for q=1:length(y_Carpet_c)
        if X_Carpet_c(y_Carpet_c(q),p)==1
            Wcurrent(q,p)=Input.retwt(p); % weight matrix
        end
    end
end
makemap(Input.retloc)
pplot (Input.retloc, 'r.')
sum(Wcurrent,2);
alaplot(Input.facloc(y_Carpet_c,:),Wcurrent,Input.retloc,'Final
Solution');

% CPLEX solution data brought back into MATLAB
cplex_loc1=Input.retloc(395,:);
pplot (cplex_loc1, 'b.');
lonlat2city(cplex_loc1,uscity)
```

Data Generation File for the hypothetical 400 collection center network

File name: Carpet_data_generation_3digitzip_A

```
% Data Generation for carpet recycling project
% % Created by Michael Bucci, Dec. 2008

function [Input] = Ryan_data_generation_3digitzip_A

% Use this code for determining collection center locations based on
% US 3 digit zip code demand weights
load ('Ryan_raw_data_3zip_A');
%     save ('Ryan_raw_data_3zip_A', 'Raw_Data');
%     [loc_3digit,pop_3digit] = uszip3('XY','Pop',uszip3('isCUS') &
uszip3('Pop') > 0);
%     Raw_Data.loc=loc_3digit;
%     Raw_Data.pop=pop_3digit;

%  ---- code to select top "x" locations by population
count=1;
for i=1:length(Raw_Data.pop)
    %if Raw_Data.pop(i)>440336      % top 200 zips by population
    if Raw_Data.pop(i)>220600       % top 400 zips by population
    %if Raw_Data.pop(i)>1           % all zip codes

        Input.retloc(count,:)=Raw_Data.loc(i,:);
        Input.retwt(count,1)=Raw_Data.pop(i);
        count=count+1;
    end
end
Input.facloc=Input.retloc;
% -------------- Code for all 877 locations  -------------
% Input.retloc=Raw_Data.loc;
% %Input.dist1=Raw_Data.dist1;
% %Input.dist2=Raw_Data.dist2;
% % If using 3 digit zip, get retwt
% Input.retwt=Raw_Data.pop;
% get possible facility locations
%[Input.facloc, Input.faczip] = uszip3('XY','Code3',uszip3('isCUS') &
uszip3('Pop') > 0);
% -------------- Code for all
 % Fixed facility costs
    Input.FC=1000000; % % Fixed cost of $1M , reference:
    % Carpet Recycling: Determine the Reverse Production System Design
% truckload cost per mile
    Input.TLcostpermile=2.32;
% truckloads per year
    %Input.TLperyear=5000; % 296M tons - current CARE network volume
    %Input.TLperyear=5051; % 300M tons
```

```
    %Input.TLperyear=10101; % 600M tons
    %Input.TLperyear=20202; % 1200M tons
    Input.TLperyear=30303; % 1800M tons
    %Input.TLperyear=40404; % 2400M tons
% distance measurement in miles
    Input.p='mi';
% secuity factor
Input.circuity=1.2;

% ---  Initial calculations ------

%normalize weights
totwt=sum(Input.retwt);
for i=1:length(Input.retwt)
    Input.retwt(i)=Input.retwt(i)/totwt;
end


% select any required facility locations
isNC=strcmp(uszip3('ST'),'NC');
isSC=strcmp(uszip3('ST'),'SC');
isGA=strcmp(uszip3('ST'),'GA');
% combine these locations
isall=logical(isNC+isSC+isGA);
[Input.facloc_SE, Input.faczip_SE] = uszip3('XY','Code3',isall);

% create transportation cost matrix
distance=dists(Input.facloc,Input.retloc,'mi');
C=distance*Input.TLcostpermile*Input.circuity*Input.TLperyear;  % cost
matrix
Input.C=[];
for i=1:size(Input.retloc,1)
    Input.C(:,i)=C(:,i)*Input.retwt(i);  % add retail wt to n x m variable
cost matrix,
      % where C(i,j) is the cost of transporting from collection center j
      % to facility location i
      %includes both retail weight and distance
end
k=ones(1,length(Input.facloc))*Input.FC; %fixed facility charge are all
equal
Input.k=k;

% data for only the SE locations for the first facility
distance=dists(Input.facloc_SE,Input.retloc,'mi');
C_SE=distance*Input.TLcostpermile*Input.circuity*Input.TLperyear;  % cost
matrix
Input.C_SE=[];
for i=1:size(Input.retloc,1)
    Input.C_SE(:,i)=C_SE(:,i)*Input.retwt(i);  % add retail wt to n x m
variable cost matrix,
```

```
      % where C(i,j) is the cost of transporting from collection center j
      % to facility location i
      %includes both retail weight and distance
end
k_SE=ones(1,length(Input.facloc_SE))*Input.FC; %fixed facility charge are
all equal
Input.k_SE=k_SE;

% ------------- economies of scale data --------------------
% NO economies of scale (i.e. constant cost for processing)
%   Input.TL_eos_bp=[2542 3367 5101 6734];
%   Input.TL_eos_cost=[3395 3395 3395 3395];

% With economies of scale
Input.TL_eos_bp=[2542 3367 5101 6734]; % Truckload breakpoints for
economies of scale
Input.TL_eos_cost=[4497 3395 3395 2572];  % processing cost per Truckload
```

Data Generation File for the CARE network

File name: Carpet_data_generation_CARE_C_non_overlap

```
% Data Generation for carpet recycling project
% % Created by Michael Bucci, Dec. 2008

function [Input] = Carpet_data_generation_CARE_C_non_overlap

% Use this code for inputting the CARE locations, and other
% information
%load ('Carpet_raw_data_CARE');

% load ('Carpet_raw_data_CARE_12'); % 12 mile radius
 load ('Carpet_raw_data_CARE_24');  % 24 mile radius
%load ('Carpet_raw_data_CARE_48');  % 48 mile radius

    %save ('Carpet_raw_data_CARE_24', 'Raw_Data', 'Input');
   % data fields: location code, zip, dist1, dist2
   % dist=distance to travel for pickup
   % field names: Input.code, Input.zip, Input.dist1, Input.dist2

% TO UPDATE VALUES USE THIS CODE HERE
% Fixed facility costs
    Input.FC=1000000;   % Fixed cost of $1M , reference:
    Input.TLcostpermile=2.32; % reference:
    % Carpet Recycling: Determine the Reverse Production System Design
% truckloads per year
    Input.TLperyear=5000; % CARE Collection volume for 2007
% secuity factor
Input.circuity=1.2;

% get possible facility locations (all 3 digit ZIP code locations)
[Input.facloc, Input.faczip] = uszip3('XY','Code3',uszip3('isCUS') &
uszip3('Pop') > 0);

% select any required facility locations
isNC=strcmp(uszip3('ST'),'NC');
isSC=strcmp(uszip3('ST'),'SC');
isGA=strcmp(uszip3('ST'),'GA');

% combine these locations
isall=logical(isNC+isSC+isGA);
[Input.facloc_SE, Input.faczip_SE] = uszip3('XY','Code3',isall);

% create transportation cost matrix
distance=dists(Input.facloc,Input.retloc,'mi');
```

```
C=distance*Input.TLcostpermile*Input.circuity*Input.TLperyear;  % cost
matrix
Input.C=[];
for i=1:size(Input.retloc,1)
    Input.C(:,i)=C(:,i)*Input.retwt(i);  % add retail wt to n x m variable
cost matrix,
      % where C(i,j) is the cost of transporting from collection center j
      % to facility location i
      %includes both retail weight and distance
end
k=ones(1,length(Input.facloc))*Input.FC; %fixed facility charge are all
equal
Input.k=k;

% data for only the SE locations for the first facility
distance=dists(Input.facloc_SE,Input.retloc,'mi');
C_SE=distance*Input.TLcostpermile*Input.circuity*Input.TLperyear;  % cost
matrix
Input.C_SE=[];
for i=1:size(Input.retloc,1)
    Input.C_SE(:,i)=C_SE(:,i)*Input.retwt(i);  % add retail wt to n x m
variable cost matrix,
      % where C(i,j) is the cost of transporting from collection center j
      % to facility location i
      %includes both retail weight and distance
end
k_SE=ones(1,length(Input.facloc_SE))*Input.FC; %fixed facility charge are
all equal
Input.k_SE=k_SE;


% NO economies of scale (i.e. zero cost for processing)
  Input.TL_eos_bp=[2542 3367 5101 6734];
  Input.TL_eos_cost=[0 0 0 0];

% ------------- economies of scale data --------------------
Input.TL_eos_bp=[1684 3367 6734]; % Truckload breakpoints for economies of
scale
Input.TL_eos_cost=[4497 3395 2572];  % processing cost per Truckload

% -------------------------------
% Code used to create Input file - using 5 digit zip and non-overlapping
% situation

% ---- Convert raw data to input data  -------------
%Input.code=Raw_Data.code;
%Input.zip=Raw_Data.zip;
%Input.dist1=Raw_Data.dist1;
%Input.dist2=Raw_Data.dist1;
```

```
% ----  other inputs  --------
% % Fixed facility costs
%    %Input.FC=75000;
%     Input.FC=1; % to avoid adding
% % truckload cost per mile
%     Input.TLcostpermile=2;
% % truckloads per year
%     Input.TLperyear=5000;
% % distance measurement in miles
%     Input.p='mi';
% % secuity factor
% Input.circuity=1.2;

% % ---  Initial calculations ------
%
% %covert zip code to XY locations
% Input.retloc= zip2lonlat(Input.zip);
% % Input.zip=[01702 27601 30201]; % for testing
%
% % If using 5 digit zip - determine demand weights for each customer
location
% zip5 = uszip5(uszip5('isCUS') & uszip5('Pop') > 0);
% zip5.count=zeros(length(zip5.Code5),1);
% for i=1:length(Input.retloc)
%    for j=1:length(zip5.XY)
%         d=dists(Input.retloc(i,:),zip5.XY(j,:),'mi');
%         if d<Input.dist1(i)
%             zip5.count(j)= zip5.count(j)+1;
%         end
%     end
% end
%
% for i=1:length(Input.retloc) % this takes care of overlap
%     Input.retwt(i)=[0];
%     for j=1:length(zip5.XY)
%         d=dists(Input.retloc(i,:),zip5.XY(j,:),'mi');
%         if d<Input.dist1(i)
%             Input.retwt(i)=Input.retwt(i)+(zip5.Pop(j,1)/zip5.count(j));
%             %Dist[i,count1]=Input.zip(i);
%             %count1=count1+1;
%         end
%     end
% end
    % for testing:
%         Input.retwt=ones(length(Input.retloc),1);
%         Input.retwt=Input.retwt*(1/length(Input.retloc));
%normalize weights
% totwt=sum(Input.retwt);
% for i=1:length(Input.retwt)
%     Input.retwt(i)=Input.retwt(i)/totwt;
% end
```

Generating CPLEX text file

File name: Carpet_textfileforCPLEX_no_eos

```
function [filedone] = Carpet_textfileforCPLEX_no_eos(Input)
%C,k,fc,fs,bp,prodvol,retwt
%  -----------------  CPLEX  -----------------------------------
%     k = n-element fixed cost vector, where k(i) is cost of NF at Site i
%           n= number of possible facility locations
%           k is a column vector
%     C = n x m variable cost matrix,
%         where C(i,j) is the cost of serving EF j from NF i
%           m= number of customer locations
%           n= number of facility locations

Ctest=Input.C;
ktest= Input.k';

% ------  create CPLEX lp file  ---------
% variable usage - 0,1 integer values
    % # = a number value representing the
    %     xy matrix in which x = NF and y = EF
    % x#y# = if facility x serves facility y
    % z# = if a facility is located at NF location (0 or 1)
    % q# = what size is each facility (total volume)
    % a#b# = constraint on z values
        % a=goes from s(1) to s(n-1)
        % b= goes from 1 to x

%[fid,mes] = fopen('test_CARE.lp','w');
[fid,mes] = fopen('carpet400_1800.lp','w');

fprintf(fid,'\\For Carpet Recycling Project\n');
fprintf(fid,'Minimize\n');

% ----  objective function
    %transportation cost
fprintf(fid,'\\transportation cost\n');
fprintf(fid,'obj: ');
counter=1;
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
%         if Ctest(i,j)==0
%         else
            fprintf(fid,'%gx%gy%g+\n',Ctest(i,j),i,j);
%         end
    end
end

    % fixed facility cost
```

```
fprintf(fid,'\\fixed facility cost\n');
for i=1:size(Ctest,1)
    if i==size(Ctest,1)
        fprintf(fid,'%gz%g\n',ktest(i),i);
    else
        fprintf(fid,'%gz%g+\n',ktest(i),i);
    end
end

% ----  add constraints  ---------------------
fprintf(fid,'\\Constraints\n');
fprintf(fid,'Subject to\n');

% each customer is served by 1 facility and all demand is met
fprintf(fid,'\\each customer served by 1 facility and all demand is
met\n');
for j=1:size(Ctest,2)
    fprintf(fid,'c%g:',j);
    for i=1:size(Ctest,1)
        if i==size(Ctest,1)
            fprintf(fid,'x%gy%g\n',i,j);
        else
            fprintf(fid,'x%gy%g+\n',i,j);
        end
    end
    fprintf(fid,'=1\n');
end

%if a facility has customers its value in the objective function =1
fprintf(fid,'\\if a facility satisfies demand, it must be open\n');
ccount=size(Ctest,2)+1;
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
        fprintf(fid,'c%g:',ccount);
        fpirintf(fid,'x%gy%g-z%g<=0\n',i,j,i);
        ccount=ccount+1;
    end
end

% ----  Integer constraints/bounds - 0 or 1  ----------
% all s#t# are >=0
    % this is the CPLEX default, no code needed
fprintf(fid,'\\integer constraints\n');
fprintf(fid,'generals\n');
for i=1:size(Ctest,1)
    for j=1:size(Ctest,2)
        fprintf(fid,'x%gy%g\n',i,j);
    end
end

% end program
```

```
fprintf(fid,'end');
fclose(fid);
filedone=1;
```

Meta-heuristic that combines constructive ADD procedure with ALA procedure

File name: Carpet_LA_model_A_eos

```
function[Input,Whitaker2] = Carpet_LA_model_eos(Input, Whitaker2);

% Hybrid construction procedure for uncapacitated facility location with
    % correlation
    % based  on Whitaker (1985) ADD/DROP formulation.
% Code by Mike Bucci, March 2008

% STEP 0: Find single facility that minimizes total distribution costs of
% the network (transportation, fixed facility, and processing costs)
costs)

% STEP 1: To add a facility... For each candidate median (each DC site not
in
% current solution) place a NF at the median location and
% allocate demand based on transportation cost only. Then add fixed
facility and processing costs

% STEP 2: reallocate customers based on all costs
% This step is repeated as the reallocation take place and facility sizes
% change until there is no improvement

% STEP 3: relocate DCs based on allocation

% Repeat step 2 and 3 until no improvement
% Return to step 1 and continue iteration until no improvement

% ------------------------------------------------------
% STEP 0
time1=clock;
[Input,TC_one_DC,X] = Carpet_initialize_eos(Input);
%TC_one_DC % for testing
time2=clock;

% Keep data on best solution for each NF level (# of facilities)
nNF=1;
Input.TCall_Whitaker(1)=TC_one_DC;
Input.TCall_Whitaker;
Whitaker2.TC_data(nNF) = TC_one_DC; % add initial cost to cost data
Whitaker2.NFloc_data{nNF,1} = Input.facloc(Input.y,:); % NF locations
Whitaker2.NF_num(nNF)=length(Input.y);
Whitaker2.X_data{nNF,1} = X; % allocation data
Whitaker2.y_data{nNF,1} = Input.y; % y values
% track eos costs
X_eos=X;
[eos_cost]= Carpet_eos_a(Input,X_eos);
```

```
Whitaker2.processing_cost_data(nNF)=eos_cost;
% track trans cost
Whitaker2.trans_cost_data(nNF)=sum(sum(Input.C.*X));
X_input=X;

nNF=nNF+1;
Xa=X;
Xin=X;
% ------------------------------------------------
% Loop for Steps 1,2,3
done5 = 0;
loopcount=1;
while ~done5
    % STEP 1
    % To add a facility
    time1=clock;
    % make adjustment for volume growth as facilities are added

    % ADD subroutine
    [Input,X,TC,W] = Carpet_add_eos(Input);
    %TC % for testing
    time2=clock;
    Input.time = Input.time + etime(time2, time1);
    Input.TCall_Whitaker(end+1)=TC;
    Input.TCall_Whitaker;
    % for TABU search
    Input.y_last = Input.y;
    length(Input.y) % show for testing
    Input.W_last=W;
    Input.X_last=X;

        % plot solution
        Wcurrent=W(Input.y,:);
        %alaplot(Input.facloc(Input.y,:),Wcurrent,Input.retloc,'W2 -
Add');
        if length(Input.y)==2;
            Input.y_last = Input.y;  % for TABU search
            Input.W_last=W;
            Input.X_last=X;
        end
        TC_NF=TC; % best solution with next NF added
        TCin=TC;
        TCin_add=TCin; % show for testing
        Xin=X;
    % ------------------------------------------------
    % Loops for STEP 2,3
    % TC = best solution so far
    done4=0;
    while ~done4 % loop until no improvement in the allocation and
location
        done3=0;
```

```
        loops=1;
        while ~done3 % loop until no improvement in the allocation
            % STEP 2: reallocate customers
            [Input,W,X,TC] = Carpet_allocate_eos(Input,X,TC); % all,
            % check for improvement
            %TC % show for testing
            if TCin > TC
                TCin = TC;
                Xin=X;
                loops=2;
                allocate_improve=1; % for testing
                Wcurrent=W(Input.y,:);
                sum(Wcurrent,2); % show for testing

%alaplot(Input.facloc(Input.y,:),Wcurrent,Input.retloc,'Whitaker2 -
Allocate Loop');
            else
                done3 = 1;
            end
            Input.TCall_Whitaker(end+1)=TC;
            Input.y;
        end % allocation loop
        TCin_alloc=TCin; % show for testing

        % STEP 3: relocate DCs based on allocation
        done_loc=0;
        TC=TCin;
        X=Xin;
        while ~done_loc
            [Input,X,W,TC] = Carpet_locate_eos(Input,X,W,TC);
            if TCin>TC  % change in TC in location solution
                loops=2;
                TCin=TC;
                Xin=X;
            else
                done_loc=1;
            end
            Input.TCall_Whitaker(end+1)=TCin;
        end
        % For tabu search
        Input.y_last = Input.y;
        Input.W_last=W;
        Input.X_last=X;

        if loops==1
            done4=1;
        end
        Wcurrent=W(Input.y,:);
        sum(Wcurrent,2); % show for testing
   end % done4 loop - Repeat step 2 (allocation) and step 3 (location)
until no improvement
```

```
    X=Xin;
    TC=TCin; %-sum(Input.k(remove_i));
    Input.TCall_Whitaker(end)=TC;

    % ------------------------------------------------
    % keep best solution from step 2 and 3
    % Also keep results of each iteration with p NFs for later analysis
    % and use in the interchange problem when p is not defined a priori
    Whitaker2.TC_data(nNF) = TC; % Total cost data
    Input.y;
    Whitaker2.NFloc_data{nNF,1} = Input.facloc(Input.y,:); % NF locations
    Whitaker2.y_data{nNF,1} = Input.y; % y values
    Whitaker2.X_data{nNF,1} = Xin; % allocation data
    Whitaker2.NF_num(nNF)=length(Input.y);

    % Get Transportation and Inventory costs
    X_input=X;
    X_eos=X;
    [eos_cost]= Carpet_eos_a(Input,X_eos);
    Whitaker2.processing_cost_data(nNF)=eos_cost;
    %calculate transportation cost
    Whitaker2.trans_cost_data(nNF)=sum(sum(Input.C.*X));


    fixed_cost_addrho=sum(Input.k(Input.y));

Whitaker2.TC_withrho(nNF)=Whitaker2.trans_cost_data(nNF)+fixed_cost_addrho
;
    %nNF
    fixed_cost_addrho=sum(Input.k(Input.y));

    Wcurrent=W(Input.y,:);
    NFsizes=sum(Wcurrent,2);
    xyz=NFsizes';
    Whitaker2.NFsizses{nNF,1} =xyz;
    nNF=nNF+1;
    if nNF-1==Input.Max_fac
        done5=1; % all possible NFs locations have a NF
    end
end % end for loop for steps 1, 2, and 3
```

Calculation of processing costs

File name: Carpet_eos_A

```
% Mike Bucci March 2008

function [inv_cost]= Carpet_eos_a(Input,X_eos)

% Economies of Scale calculation for
% carpet recycling project
%
% Input.y provides facilities in current solution
% X_input provides allocation

% % get facility size and processing costs
inv_cost=0;
for p=1:length(Input.y)
    DC_step=(X_eos(Input.y(p),:).*Input.retwt');
    DC_size=sum(DC_step)*Input.TLperyear;
    if DC_size <=Input.TL_eos_bp(1)
        inv_cost=inv_cost+Input.TL_eos_cost(1)*DC_size;
    elseif DC_size <=Input.TL_eos_bp(2)
        inv_cost=inv_cost+Input.TL_eos_cost(2)*Input.TL_eos_bp(2);
    elseif DC_size <=Input.TL_eos_bp(3)
        inv_cost=inv_cost+Input.TL_eos_cost(3)*DC_size;
    elseif DC_size <=Input.TL_eos_bp(4)
        inv_cost=inv_cost+Input.TL_eos_cost(4)*Input.TL_eos_bp(4);
    else
        inv_cost=inv_cost+Input.TL_eos_cost(4)*DC_size;
    end
end
```