INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

SUPPLY CONNECTED LOCATION-ALLOCATION PROBLEM

By
GÜLCAN N. YEŞİLKÖKÇEN

A Thesis
Submitted to the School of Graduate Studies in
Partial Fulfillment of the Requirements
for the Degree
Doctor of Philosophy

McMaster University

SUPPLY CONNECTED LOCATION-ALLOCATION PROBLEM

DOCTOR OF PHILOSOPHY (1997) McMaster University
(Management Science/Systems) Hamilton, Ontario

TITLE: Supply Connected Location-Allocation Problem

AUTHOR: Gülcan N. Yeşilkökçen
M.Sc. Bilkent University
B.Sc. Bilkent University

SUPERVISORY COMMITTEE: Prof. George O. Wesolowsky (Chairman)
Prof. Robert F. Love
Prof. George Steiner

NUMBER OF PAGES: xxiii, 241

Abstract

This study introduces a new model for a location distribution system, which is called the *Supply Connected Location Allocation Problem* (SCLAP). The problem involves locating p facilities with respect to n demand points and a supply plant. The supply plant originates all material in the system and distributes it to facilities along a route to be determined. Facilities then distribute the material to demand points via direct shipments. The problem minimizes the cost of shipping material from supply plant to all demand points via facilities.

A description of the model is given and its subproblems are discussed in relation to well-known models in the literature. The problem is then investigated in three avenues: First, some special cases on which the problem is efficiently solvable are identified. For this, simple structured networks (e.g. a line and a tree) are considered. A dynamic programming solution procedure with polynomial time complexity is developed for the case on a line and it is extended to a special case of tree networks.

Second, the problem is considered on general networks and a single assignment branch-and-bound algorithm is proposed to solve it. The algorithm is tested on randomly generated networks for relatively small sizes of problem instances. Discussions on the computational results are given with respect to computation time and the size of the branch and bound tree.

Next, local search type heuristics are discussed briefly and two heuristic approaches are developed. Two versions of algorithms are constructed for each of those heuristics and they are tested on randomly generated networks, with respect to computation time and closeness to optimality (when optimal solutions were available)

or to closeness to the best solution found. Finally, the study is concluded with recommendations for future research.

Preface

Findings of Chapters 2 and 3 of this thesis have already been prepared for publication by the candidate, co-authored by Professor George O. Wesolowsky. The first paper which includes the findings in Chapter 2 has been submitted for publication to *Location Science* in November 1996, and the second paper which discusses the findings in Chapter 3 is submitted to the *ISOLDE VII Special Issue* of *Location Science* in December 1996. Both papers are currently under the refereeing process. All the research and the findings in those papers were done by the candidate under the supervision of Professor George O. Wesolowsky.

Acknowledgements

I would like to thank Professor George O. Wesolowsky, my supervisor, for his invaluable guidance and support throughout the development of this research.

I wish to extend my thanks to the members of my supervisory committee, Professor Robert F. Love and Professor George Steiner, for their helpful comments and suggestions on this thesis.

Finally, I wish to thank my family for their continuous love and support that reach me from thousands of miles away.

to my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Review

With businesses becoming larger and ever more complex, the problems that need to be modeled, formulated and solved, also grow in their size and complexity. Real life applications, in general, involve many aspects of a problem to be optimized simultaneously, and therefore hybrid formulations of two or more "classical" types of problems are common. Aside from the growing requirements of solutions to large and complex real-life problems, advances in computer technology allow such problems to be relatively more tractable. One example of such hybrid models is the Location-Routing Problem (LRP). Consider a case where a factory has to ship its product to warehouses, from where the goods will be distributed to customers. The problem to be solved in this case, may involve determining the number of warehouses to be opened, their locations, and the set of customers that each warehouse will serve, as well as the routes for shipment for goods from factory to warehouses or from warehouses to customers. LRPs therefore optimize routing, location, and allocation simultaneously. In effect, LRPs incorporate the classical Location-Allocation and Routing problems.

We will study each of these classical types of problems and discuss their relationship to LRPs, before introducing the problem we study in this thesis.

1.1 Location-Allocation Problems

Location Problems arise in a variety of contexts. Location of stores, retail outlets, warehouses, or factories is a crucial factor in costs and profitability of a company in the private sector. Locations of emergency facilities, fire stations, ambulances, and hospitals are important concerns for public sector entities. For example, even individuals, when deciding to buy a new house, are faced with location problems. Different aspects of a house's location may increase or decrease its monetary value. Closeness to amenities such as grocery stores, schools, parks, or distance from a garbage dump site might add or deduct significant amounts to and from the value of the property just as the physical aspects of the house would do.

A typical facility location problem addresses the questions of where to locate, how many facilities to locate, etc. In some cases where there is more than one facility to locate, the interaction between the customers and the facilities may not be known a priori, and the decision to allocate customers to one of the facilities has to be made simultaneously with the locational decisions. Those kinds of problems are referred to as *location-allocation* problems (Love, Morris, and Wesolowsky 1988). In most cases in the literature, however, location problems also refer to both pure location and location-allocation problems. In this section of the thesis we focus our attention on location-allocation problems and whenever the term "location problem" is used it is assumed that the allocation component has to be determined as well.

Costs related to transportation constitutes an important component in many firms' operations. Locational decisions are a major factor affecting these transportation costs. Therefore, in many location problems, the typical measure of quality of service is some function of distances between facilities and customers.

Quantitative approaches to formulating and solving location problems date back to early seventeenth century. Fermat (1601–1655) proposed the question: "given three points in the plane, find a fourth point such that the sum of its distances to the three given points is minimum". Wesolowsky (1993) gives a detailed discussion of the origins of the problem and contributions to it in the literature from a historical perspective. The problem, often called the *Weber Problem* (due to Weber 1909) or the *Euclidean Minisum Problem*, has attracted many researchers from many fields and numerous generalizations have been constructed. Today, the broader area of location analysis involves many researchers from a wide variety of academic fields which span computer science, economics, engineering, geography, management science, mathematics, operations research, and urban planning and regional science.

As pointed out in the editorial of the first issue of the *Location Science* journal (Church, Current, and Eiselt 1993), there are many factors which create this tremendous interest in location analysis:

> '...First, location decisions are frequently made at all levels of human organization from individuals and households to firms, governments, and international agencies. Second, such decisions are often strategic in nature: that is, they involve significant capital resources and their

> economic effects are long term in nature. Third, they frequently impose economic externalities. Such externalities include economic development, as well as pollution and congestion. Fourth, location models are often extremely difficult to solve, at least optimally. Even some of the most basic models are computationally intractable for all but the smallest problem instances. In fact, the computational complexity of location models is a major reason that the widespread interest in formulating and implementing such models did not occur until the advent of high speed digital computers. Finally, location models are application specific. Their structural form, "the objectives, constraints and variables", is determined by the particular location problem under study. Consequently, there does not exist a general location model that is appropriate for all, or even most applications.'

As reflected by the diversity of researchers from different fields, location problems find applications in many different areas. While hospitals, factories, warehouses, retail outlets, fire stations etc., are typical examples, applications of location problems are also found in computer network design, circuit design, automated manufacturing systems, exploratory oil drills, telecommunications, as well as forestry, water irrigation systems, etc.

Location decisions are critical at almost every level of a private or a public service enterprise. They range from location of relatively inexpensive facilities like computer workstations, or tools in a workshop, to capital intensive investments like factories, or major hospitals. They may involve location of a single facility or multiple

facilities. Facilities may also interact with each other (e.g. a plant-warehouse-retail outlet network), or provide comparable service only to a given set of customers or demand points. Location decisions may involve one or more sometimes conflicting objectives. For example, not all facilities are "desirable". In some cases facilities like garbage dump sites, nuclear reactors, and waste treatment plants attract public resistance because people do not want them near their homes. Therefore, they have to be placed in remote areas; however this in turn results in greater transportation costs. Additionally, location problems may be posed on a plane, on a transport network, or sometimes the facilities are required to be located on predefined sets of discrete points.

Location problems can be classified in a number of ways, depending on many different aspects of the models, as discussed above. The classifications may be done according to the location space used (planar vs network vs discrete location problems), number of facilities to be located (single vs multiple), nature of inputs (deterministic vs probabilistic), capacity restrictions on facilities (capacitated vs uncapacitated) and so on. In the first chapter of the book by Daskin (1995), a taxonomy of location problems and models are provided based on such classifications. Similar taxonomies have been given by Brandeau and Chiu (1989) and Krarup and Pruzan (1990). In (Brandeau and Chiu 1989), over fifty different location problems have been reported. A Bibliography of Location Problems by Domschke and Drexl (1985) includes well over 1500 entries. Additionally, the book edited by Drezner (1995) provides a state of the art review of some of the location problems. At the end of the overview section of (Drezner 1995) a list of fifteen books, sixteen review articles and fourteen special volumes on location analysis that have been published

since about 1980 is provided. Some of those include reviews and other books on location analysis like (Francis and Goldstein 1974), (Francis, L. F. McGinnis, and White 1992), (Thisse and Zoller 1983), and (Love, Morris, and Wesolowsky 1988).

As it is already pointed out, there is a broad family of location problems. The subject of this thesis falls into the category of *network* and *discrete location* problems.

1.2 Network and Discrete Location Problems

Network and discrete location problems arise when the location space is a network or a set of discrete points. In *planar* or *continuous location models*, facilities can be located anywhere on the plane; the demands may also occur anywhere on the plane or they may occur at specific points on the plane. In network location models, however, the demands are assumed to occur only on a network or graph composed of nodes and links. Facilities can be located only on the nodes or the links of the network. Networks of highways, railways, pipeline systems, computer communication networks are examples of such networks used in location problems. In discrete location models, demands occur at specific points, and facilities are to be located on some set of discrete points serving as candidate sites. Chhajed, Francis, and Lowe (1993) provide further discussions on discrete and continuous models and their differences. The reader is also referred to books by Hurter and Martinich (1989) and Love, Morris, and Wesolowsky (1988) for discussions on planar location models. For a focus on network and discrete location models, the reader may consult the books by Handler and Mirchandani (1979), Daskin (1995) and the book edited by Mirchandani and Francis (1990).

Among the innumerable extensions and generalizations of the location problems, a few of the basic models (e.g. p-median, p-center, uncapacitated facility location, and covering problems) play an important role, not only because they find variable applications in real life problems, but also because they tend to appear as subproblems or components of other generalizations (Mirchandani and Francis 1990; Daskin 1995). We will discuss two of those models in detail, namely the p-median and the uncapacitated facility location. This is because a special form of the problem we study reduces to the p-median problem under certain conditions, and an immediate extension, where the number of facilities to be located are not given and fixed costs are associated with each facility site, includes uncapacitated facility location as a special case.

1.2.1 The p-Median Problem

The p-median problem involves finding locations of p facilities among m candidate sites to serve n demand points. The problem was first formulated by (ReVelle and Swain 1970). Facilities provide comparable service, therefore each demand point receives service from the nearest facility. The cost of serving a demand point is directly proportional to the amount of demand at that point and the distance between the demand point and the nearest facility. We say a demand point i is *allocated* to a facility j, if it receives service from that facility. The problem can be formulated as follows:

Let the *parameters* be denoted by

w_i = weight (demand) at node i

d_{ij} = distance between demand node i and candidate site j

p = number of facilities to locate

n = number of demand points

m = number of candidate sites for location of facilities

and the *decision variables* be denoted by

$$Y_{ij} = \begin{cases} 1 & \text{if demand point } i \text{ is allocated to a facility at site } j \\ 0 & \text{otherwise} \end{cases}$$

$$Z_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \\ 0 & \text{otherwise} \end{cases}$$

Y_{ij}'s are termed the *allocation variables* and Z_j's are termed the *location variables*. The formulation of the problem is now as follows:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_i d_{ij} Y_{ij} \qquad (1.2.1.a)$$

$$\text{subject to} \quad \sum_{j=1}^{m} Y_{ij} = 1 \qquad i = 1, \ldots, n \qquad (1.2.1.b)$$

$$\sum_{j=1}^{m} Z_j = p \qquad (1.2.1.c)$$

$$Y_{ij} - Z_j \leq 0 \qquad i = 1, \ldots, n, \quad j = 1, \ldots, m \qquad (1.2.1.d)$$

$$Y_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n, \quad j = 1, \ldots, m \qquad (1.2.1.e)$$

$$Z_j \in \{0, 1\} \qquad j = 1, \ldots, m \qquad (1.2.1.f)$$

The constraints (1.2.1.b) requires that each demand node is allocated to exactly one facility. Constraint (1.2.1.c) states that there will be exactly p facilities located. Constraint (1.2.1.d) ensures that a demand point i can only be allocated to a facility at site j if there is a facility located at that site. Constraints (1.2.1.e) and (1.2.1.f) are the standard integrality constraints. Based on those constraints, the

objective function (1.2.1.a) minimizes the total transportation cost (demand-weighted distance) from each demand node to the nearest facility.

Constraints (1.2.1.d) that link the location and the allocation variables can be replaced by constraints of the form:

$$\sum_{i=1}^{n} Y_{ij} - nZ_j \leq 0 \quad . \quad j = 1, \ldots, m \quad (1.2.1.d')$$

to get an equivalent formulation of the problem (Efroymson and Ray 1966).

The former set of constraints with (1.2.1.d), are called the *stronger* version, while the latter with (1.2.1.d') are called the *weaker* version. The weaker version of the constraints reduces the total number of constraints; however, the linear relaxation of the weaker version obtained by replacing the integrality constraints (1.2.1.e) and (1.2.1.f) by nonnegativity constraints, results in weaker lower bounds for the optimal solution value than the stronger version. The linear relaxation of the stronger version provides what ReVelle (1993) calls "integer friendly" solutions. That is, the results obtained from the relaxation consists of mostly integer variables.

The p-median formulation given above assumes that there is a finite number of candidate sites for facilities to be located. This is also true in the context of networks. Even when the facilities are allowed to be located anywhere on the network. Hakimi (1965) has shown that *for the p-median problem, there exists at least one optimal solution such that all facilities are located on the nodes of the network.* This property, which is called the *node-optimality property*, allows us to formulate the p-median problem on networks as an Integer Program, by selecting the set of nodes of the network as candidate sites for the facilities.

The node-optimality property of the p-median problem has been extended

to several variants and generalizations of the problem. Handler and Mirchandani (1979), Mirchandani and Odoni (1979), and Mirchandani (1990) discuss such extensions with other references to the probabilistic demands, oriented networks, and various transportation cost measures as well as multidimensional networks and multiple commodities.

The p-median problem is $\mathcal{NP}$-Hard on general networks (Kariv and Hakimi 1979; Garey and Johnson 1979). Therefore, enumeration type approaches in solving the problem may require unrealizable computational efforts for large instances of the problem. Thus, researchers have developed several different approaches for solving the p-median problem. Those can be classified as (*i*) graph-theoretic, (*ii*) heuristic, and (*iii*) mathematical programming approaches.

One traditional approach to solving such problems on networks is to identify simpler network structures on which the problem is efficiently solvable. *Tree* networks are one of those network structures that allow efficient solvability for many of the "difficult" combinatorial optimization problems. Kariv and Hakimi (1979) present a dynamic programming type algorithm for the problem on trees. The computational complexity of their algorithm is $O(p^2n^2)$, which is second order polynomial on the number of facilities and the number of nodes of the tree.

Goldman (1971) gives a very simple and elegant algorithm of order $O(n)$ for the 1-median of a tree. His algorithm is based on the observation that the 1-median of a tree is contained in a subtree of the tree which includes half or more of the total demand in the system. The algorithm performs "folding" operations on demands at tip nodes (i.e. it adds the amount of demand of a tip node to the amount of the demand at the adjacent node and deletes the tip node and its incident arc).

The procedure is continued until a node of the new tree contains half or more of the total demand. This node is identified as the 1-median.

For 2-medians of a tree, Mirchandani and Oudjit (1980) give an *arc-deletion method* of order $O(n^2)$. The main idea is the partitioning of the tree into two subtrees and solving 1-median problems on each of the subtrees. Since there are (n-1) arcs on a tree, (n-1) partitions are considered, and the one with the minimum objective value gives the solution. In addition to efficient algorithms for the p-median problem, Dearing, Francis, and Lowe (1976) discuss some convexity properties of tree networks, which partly explain why many of the difficult combinatorial optimization problems are efficiently solvable on trees. Tansel, Francis, and Lowe (1983) provide a detailed survey of several classes of location problems on networks with an emphasis on the works utilizing the underlying network structure.

The heuristic approaches to solving the p-median problem include the "myopic" approach (Kuehn and Hamburger 1963), the "node partitioning" algorithm (Maranzana 1964), and the "node substitution" algorithm (Teitz and Bart 1968) as well as several heuristic branch-and-bound schemes. The references cited above are essentially the earliest studies of the *greedy* or *myopic* heuristics, *neighborhood search* and *exchange* heuristics for the p-median problem, respectively. The greedy algorithm falls into the broader category of *construction* algorithms, while neighborhood search and exchange heuristics fall into the category of *improvement* algorithms (Golden, Bodin, Doyle, and Stewart 1980). The construction algorithms start from scratch and try to build good solutions based on previously made location and allocation decisions. The improvement algorithms on the other hand, start with a solution at hand and try to improve the solution by moving either the locations of facilities or

the allocations of demand points at each iteration.

The *myopic algorithm* for the p-median problem starts with no facilities located. In the first step, it locates a 1-median which can be found by complete enumeration in polynomial time. In the second step, it tries to locate another facility with respect to all demand points and the first facility whose location is already fixed. The procedure continues in this fashion until all p facilities are located. At iteration k of the algorithm the locations of $k-1$ facilities are already fixed and the location of the k-th facility is determined. This facility is located on the candidate site j^* which minimizes the cost of allocating all demand points to the previously located $k-1$ facilities and the k-th facility. At each iteration of the algorithm when a new facility is located on an available site, some of the demand points that were allocated to previously located facilities now become closer to the new facility. Hence there is an improvement on the objective function value. The amount of this improvement is maximized at each iteration. Although the solution found by this algorithm may not be optimal, it is appealing because of its simplicity of implementation. Additionally, it can find use in real life problems when a certain number of facilities is already located and another facility is being considered for location. If the problem is to locate only one additional facility this approach will find the optimal solution for the location of the additional facility with respect to other facilities already existing. Note, however, this is not an optimal solution for the p-median problem with the augmented number of facilities.

Consider the structure of the solution to the p-median problem. When all the facility locations are fixed (optimal or not), the demand points are allocated to the nearest facilities. This is because the facilities are assumed to have unlimited capacity

and the objective is to minimize the total weighted distance. This, in effect, partitions the demand point set into p subsets, such that those demand points that are in the same subset are allocated to the same facility. Those nodes that are in the same subset are referred to as the *neighborhood* of the facility to which they are allocated. Within each neighborhood, one can locate its corresponding facility efficiently by solving a 1-median problem. The *neighborhood search heuristic* starts with an arbitrary set of p facility sites and determines the allocation sets (neighborhoods) for those sites. Then it finds the optimal 1-median within each allocation set. If any of the facility locations change, then the demand points are reallocated to facilities to form new neighborhoods. Similarly, if any of the allocation sets change, then the facilities are relocated and so on. To improve the solution obtained by the myopic algorithm, one can use the facility sites obtained from the myopic algorithm as the starting point for the neighborhood search heuristic. The essential limitation of the neighborhood search heuristic is that, at every iteration the relocations are based on the "local" conditions, i.e. only on the neighborhood of the given facility. As a result, some of the relocation possibilities which do not improve the local conditions but may, in fact, be beneficial in the "global" sense, are being dismissed.

In the *exchange heuristic*, the facilities are allowed to be located outside their neighborhoods as well. The algorithm again initializes with an arbitrary set of p facility locations. At each iteration the best replacement site for a given facility is identified. If the objective function is reduced, the replacement site and the current site are exchanged. The procedure is repeated until all of the facilities are considered and none of the relocation movements can result in an improvement of the objective function. Note that, one can also select to relocate a facility to a new site immediately,

when such relocation results in an improvement on the objective function, instead of selecting the best replacement site at each iteration.

The mathematical programming approaches to solving the p-median include linear relaxation (ReVelle and Swain 1970; ReVelle 1993), branch-and-bound algorithms (Khumawala 1972), Dantzig-Wolfe decomposition (Garfinkel, Neebe, and Rao 1974), variable upper bounding scheme (Schrage 1975), Lagrangian relaxation (Narula, Ogbu, and Samuelson 1977; Cornuejols, Fisher, and Nemhauser 1977), and the linear programming dual (Galvão 1980). Handler and Mirchandani (1979) provide detailed discussions on some of these approaches.

1.2.2 The Uncapacitated Facility Location Problem

In all the models considered so far the number of facilities to be located was fixed to a number p beforehand. By constraining the number of facilities we implicitly incorporate the facility construction costs into the model, which may be imposed by a budgetary constraint. However, by doing so we also assume that the facility location costs are identical on each candidate site. As discussed in (Daskin 1995), while such assumptions may be applicable in the public sector, where the cost bearers and the beneficiaries of the facilities are two different groups, and costs and benefits are incomparable, they may not be as useful in the private sector, where a single actor pays for the costs and realizes the benefits at the same time with comparable costs and benefits measured in monetary terms.

In those cases the problem objective is to minimize the total facility location cost and the transportation costs. The problem is called the *uncapacitated facility location problem* or the *simple plant location problem* in the literature. The

model utilizes

the parameters

c_{ij} = transportation cost between demand point i and candidate site j

f_j = fixed cost of location at candidate site j

decision variables

Y_{ij} = fraction of demand at node i that is served by a facility at site j

$$Z_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \\ 0 & \text{otherwise} \end{cases}$$

The formulation of the problem is given as follows:

$$\text{Minimize} \quad \sum_{j=1}^{m} f_j Z_j + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} Y_{ij} \qquad (1.2.2.a)$$

$$\text{subject to} \quad \sum_{j=1}^{m} Y_{ij} = 1 \qquad i = 1, ..., n \qquad (1.2.2.b)$$

$$Y_{ij} \leq Z_j \qquad i = 1, ..., n, \quad j = 1, ..., m \qquad (1.2.2.c)$$

$$Y_{ij} \geq 0 \qquad i = 1, ..., n, \quad j = 1, ..., m \qquad (1.2.2.d)$$

$$Z_j \in \{0, 1\} \qquad j = 1, ..., m \qquad (1.1.4.e)$$

The objective function (1.2.2.a) minimizes the total set-up cost and the transportation cost. Constraints (1.2.2.b) ensure that all demand at each demand point is served. Constraints (1.2.2.c) state that a demand point i receives service from a facility j only if there is a facility located at that site. Constraints (1.2.2.d) are the nonnegativity constraints for the allocation variables, and the constraints (1.2.2.e) are the integrality constraints for the location variables. Here again, it can be shown that the allocation variables will take integer values in the optimal solution.

Note the resemblance between the uncapacitated facility location problem and the p-median problem we discussed before. The only difference between the two models is the inclusion of set-up costs for facilities and the exclusion of the requirement that a fixed number of facilities has to be located. Because of the apparent similarities between the two models, the solution approaches to the uncapacitated facility location problem are very similar to those used for the solution of the p-median problem.

The uncapacitated facility location problem has been studied extensively and there is a vast amount of literature to date. Recently ReVelle and Laporte (1996) proposed several extensions to the basic plant location problem under three major categories including (i) new objectives, (ii) multiple product/multiple machine plant locations, and (iii) plant location with spatial interactions. Krarup and Pruzan (1983) provide a survey of the uncapacitated facility location problem, which also explores the origins (first formulations) of the problem. They conclude that the problem has been formulated independently by Balinski and Wolfe (1963), Kuehn and Hamburger (1963), and Stollsteimer (1963). Cornuejols, Nemhauser, and Wolsey (1990) discuss the problem in detail and provide a summary of several approaches that were developed to solve the problem.

The uncapacitated facility location problem has been used in a variety of applications ranging from bank account location (Cornuejols, Fisher, and Nemhauser 1977), to location of off-shore drilling platforms (Rosing and Odell 1978; Hansen, Pedrosa Filho, and Ribeiro 1992) and many other applications.

The uncapacitated facility location problem is shown to be NP-Hard in general (see Cornuejols, Nemhauser, and Wolsey 1990 for a proof by reducing *node*

cover problem [1] to the uncapacitated facility location problem). The problem has been shown to be polynomially solvable on tree networks (Kolen 1983), also when a classical economic lot size problem is formulated as an uncapacitated facility location problem (Krarup and Bilde 1977).

A variety of heuristic approaches to solving the uncapacitated facility location problem has been developed in the literature. The construction type algorithms include the ADD and DROP heuristics. These are also the greedy type heuristics in which a maximum improvement is sought for at each iteration. The ADD heuristic starts with no facilities open and locates a facility at each step until opening another facility no longer improves the objective. Conversely, the DROP heuristic starts with facilities open at *all* candidate sites and closes one facility at a time until closing another facility does not improve the objective.

As in the case of the p-median problems, improvement heuristics, like the exchange algorithms, can be developed to improve on the results obtained from any of the greedy heuristics. Kuehn and Hamburger (1963) developed a heuristic consisting of two subroutines. The first subroutine is basically the ADD heuristic and the second subroutine is an exchange heuristic which they call the "bump and shift" routine. The solution obtained from the "add routine" is first analyzed and any facility that has become uneconomical because of presence of other facilities that are located subsequently, is eliminated (bumped). This provides the starting feasible solution for the exchange heuristic in which each open facility is considered for relocation to an available candidate site. Cornuejols, Fisher, and Nemhauser (1977) gave bounds for

[1] **Node Cover Problem:** Given a graph G and integer k, does there exist a subset of k nodes of G that cover all the arcs of G? (Node v is said to cover arc e if v is an end point of e). This problem is $\mathcal{NP}$-Complete (Garey and Johnson 1979).

the greedy and exchange heuristics on their worst-case performance.

The solution approaches to solving the uncapacitated facility location problem optimally include (i) Bender's decomposition applied to the Mixed Integer Programming formulation of the problem (Balinski and Wolfe 1963; Balinski 1965; Magnanti and Wong 1981) (ii) Branch and Bound algorithms which utilize the fact that the allocation variables need not be constrained to integrality (Efroymson and Ray 1966; Spielberg 1969; Khumawala 1972). ReVelle and Swain (1970) observed that the linear relaxation of the problem that uses the "strong" version (described in Section 1.2.1) resulted in integral solutions very often, and therefore it provided a better bound on the objective function. Although this significantly reduces the size of the branch and bound tree, strong linear programming relaxation is quite a bit larger in size, and this makes it difficult to solve it directly by simplex method. Therefore, many researchers have focused on special purpose algorithms for solving a strong linear programming relaxation of the uncapacitated facility location problem. Several different approaches are studied by Garfinkel, Neebe, and Rao (1974), Schrage (1975), Guignard and Spielberg (1977), and Cornuejols and Thizy (1982). Dual based approaches are proposed by Bilde and Krarup (1977) and Erlenkotter (1978). Erlenkotter's DUALOC algorithm has been very effective and appears to outperform all existing algorithms. Additionally, Lagrangian approaches have been studied by Geoffrion (1974), Narula, Ogbu, and Samuelson (1977), Cornuejols, Fisher, and Nemhauser (1977).

1.3 Routing Problems

Like location problems, routing problems arise in a variety of contexts. An example is package and mail delivery. In those cases the pick-up and delivery functions are the basis of the operation, and the routing problem is especially important. Routing problems also find applications in advanced manufacturing systems, computer network design, VLSI design, cutting wall papers, clustering data array, etc.

Typically, a routing problem is to find one or several routes so that the traveller can start from a specific point and visit a set of given points with respect to some criteria. In general, costs related to delivery time or length of the route are the major components of the criteria for service quality. It is generally assumed that the travel time between two points is directly related to the distance between them. Therefore, in many routing problems, the typical measure of quality of service is some function of distances between points (which are also referred to as "customers" or "cities" in the literature).

Earliest formulations of routing problems date back to eighteenth century, when both Euler (1759) and Vandermonde (1771) discuss the problem of *knight's tour* (Biggs, Lloyd, and Wilson 1976). Starting from a corner of the board, the problem is to visit all 64 squares on a chess board exactly once with legal knight moves at each step.

A *cycle* in a graph is defined to be the set of vertices of the graph such that it is possible to move from vertex to vertex along edges of the graph so that all vertices are encountered exactly once. The *Hamiltonian cycle* problem, named after

the Irish mathematician Sir William Rowan Hamilton, seeks to find such a cycle that visits all the vertices of the graph. Kirkman (1856) gave a sufficient condition for the existence of Hamiltonian cycles in a polyhedral graph. He also showed that a polyhedron on an odd number of vertices, in which each face has an even number of edges, cannot have a Hamiltonian cycle. The reader is referred to (Biggs, Lloyd, and Wilson 1976) and (Lawler, Lenstra, Kan, and Shymoys 1985) for a detailed discussion on the early history of routing problems.

Routing problems attracted the attention of researchers from a wide range of backgrounds. An impressive number of variations of routing problems has been formulated and studied in the literature. In the last half of the twentieth century, advances in technology and demands for better and more efficient systems in all areas of industry, from manufacturing to service, gave rise to many applications, and consequently new formulations, of a wide variety of routing problems.

At the very core of the class of routing problems lies the *Traveling Salesman Problem* (TSP). We can, in fact, say that the TSP is the prototype of routing problems. Therefore, in the next section, we will discuss TSP and existing research up to date related to it. We will mention some variants of the problem only briefly, except the *Delivery Man Problem* (DMP) will be discussed in more detail, because a variant of the DMP appears as a subproblem of the problem we study in this thesis.

1.3.1 The Traveling Salesman Problem (TSP)

Given a graph G with n vertices, the traveling salesman problem involves finding a tour that starts from a given vertex and visits all vertices exactly once before

returning to the starting vertex. Clearly, the existence of a Hamiltonian cycle has to be guaranteed in this problem. A simple variant of this problem is to allow visiting vertices more than once, and ask to find the minimal tour that visits all vertices at least once. It is simple to show that the problem of visiting all vertices at least once can be transformed to visiting all vertices exactly once (Garfinkel 1985) by simply replacing the distance matrix with shortest path lengths between any two points, instead of edge lengths. Traditionally, points to be visited in a TSP are called "cities" in the literature, a designation related to the early history of the problem (Hoffman and Wolfe 1985). We will use similar terminology in this section.

Menger (1930) presented a problem in connection with a "new definition of curve length" that proposed:

> "The length of a curve be defined as the least upper bound of the set of all numbers that could be obtained by taking each finite set of points of the curve and determining the length of the shortest polygonal graph joining all the points."

The problem of finding the shortest path joining all of the finite set of points, whose pairwise distances are known, is called the *messenger problem* by Hoffman and Wolfe (1985). They also identify it as a more direct precursor of the TSP, in which edge lengths play a prominent role. It differs from the TSP by not requiring a cycle, but rather a path. Hoffman and Wolfe (1985) provide some discussion on the first use of the term "Traveling Salesman Problem" in mathematical circles, as well as its possible origins from a book published in 1932 in Germany (Voigt 1831; Müller-Merbach 1983).

The TSP has been studied by many researchers from a wide variety of

areas, such as operations research, management science, mathematics, computer science, physics, and so on. There is a vast amount of literature on it. The book edited by Lawler, Lenstra, Kan, and Shymoys (1985) focuses solely on this problem with discussions on every aspect of it, from its history to motivation and modeling, to well-solved special cases, as well as exact and heuristic solution methods. More recently Jünger, Reinelt, and Rinaldi (1994) presented a survey article on the problem. Reinelt (1994) published a book on TSP with an emphasis on the question of computation, i.e. finding good or acceptable tours for large scale problems in shorter times. He provided performance comparisons for many exact and heuristic algorithms developed in the literature for solving TSPs.

Dantzig, Fulkerson, and Johnson (1954) were the first to solve a sizable TSP. Since the 48-city problem solved in that seminal paper, a number of problems have been reported to be solved to optimality: a 120-city problem (Grötschel 1980), a 318-city problem (Crowder and Padberg 1980), a 532-city problem (Padberg and Rinaldi 1987), a 666-city problem (Grötschel and Holland 1991) and a 2392-city problem (Padberg and Rinaldi 1991). Of course, studies on larger examples of the TSP have been reported, but those mentioned above are the ones that were proven to be solved to optimality. With developments of heuristics, many large scale problems were solved to find a tour within certain upper and lower bound values. However, many of those solutions have not yet been proven to be optimal. This progress was made possible by development of mathematical theory and efficient algorithms, as well as advances in computer technology.

Despite these achievements, many aspects of TSP are still open to question. Therefore, the problem attracts new research continually. Since TSP

algorithms also appear as components for many other combinatorial optimization problems, efficient solution approaches to TSP might open up avenues for better procedures to solve "harder" problems. Secondly, most algorithms that solve large scale TSPs are not robust in the sense that solution times vary strongly for different problems with the same number of cities.

The TSP can be formulated as a Linear Integer Program as follows:

Let x_{ij} be the 0-1 variable indicating whether or not the salesman goes directly from city i to city j, and c_{ij} be the corresponding distance. Then we can formulate the problem as follows:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad (1.3.1.a)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1,..,n \qquad (1.3.1.b)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad j = 1,..,n \qquad (1.3.1.c)$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \quad S \subset \{1,....,n\} \text{ and } S \neq \emptyset \qquad (1.3.1.d)$$

The objective, (1.3.1.a), is to minimize the tour length subject to constraints given in (1.3.1.b-d). Constraints (1.3.1.b) and (1.3.1.c) ensure that only one city can be visited before a given city and only one city can follow a given city on the tour. Constraints (1.3.1.d) are called the *subtour elimination constraints* which ensure that no subtours are allowed in the solution, i.e. the TSP tour will consist of a single tour that visits all cities. The above formulation without the subtour elimination constraints describes the well-solved *assignment problem.* Solution to an assignment problem may contain many subtours rather than a single tour visiting all cities. Thus, the assignment problem is a relaxation of the TSP. Of course (1.3.1.d)

represent a large number of constraints ($2^n - 2$ to be exact), and therefore this formulation is inefficient in its complete form. However, it has at least one good characteristic of having a well-solved relaxation, namely the assignment problem.

A more compact version of constraints (1.3.1.d) is proposed by Miller, Tucker, and Zemlin (1960) allowing TSP to be modeled as a mixed integer program in $O(n^2)$ constraints.

As is the case with many hard problems, solution approaches to the TSP involve many heuristics as well as exact solution procedures. Exact solution approaches include the use of *cutting planes*, branch-and-bound, and dynamic programming procedures. Combinations of cutting planes and branch-and-bound have been very successful. On the other hand, due to its enormous storage requirements, dynamic programming can only solve relatively small problems.

Because the TSP is a major problem which is, almost always, used as a basis to develop and test new ideas of heuristics for combinatorial optimization problems, there is an incredibly large amount of research on heuristic approaches to TSP (Reinelt 1994). Those approaches can be divided into two major categories; (*i*) construction heuristics, (*ii*) improvement heuristics. More recently, however, new types of heuristics including Simulated Annealing, Tabu Search, Neural Networks, or Genetic Algorithms have also been applied to TSP.

Construction heuristics determine a tour according to some construction rule, but do not try to improve upon this tour; a tour is successively built and parts already built remain, in a certain sense, unchanged throughout the algorithm. For a detailed discussion on construction heuristics and computational results, the reader is referred to Golden and Stewart (1985), Arthur and Frendeway (1985), Johnson

(1990) and Bentley (1992).

There are many versions of construction heuristics applied to TSP. However, they can be roughly placed in four major categories (Reinelt 1994): (*i*) nearest neighbor heuristics, (*ii*) insertion heuristics, (*iii*) heuristics based on spanning trees, and (*iv*) savings heuristics.

In *nearest neighbor heuristics*, the salesman starts at some city, then visits the city nearest to the starting city. From there, he visits the nearest city that was not visited so far, etc. until all cities are visited, and the salesman returns to the starting city. Rosenkrantz, Stearns, and Lewis (1977) show that no constant worst case can be given for this simple heuristic, that is, no constant upper bound can be calculated on the objective value of solutions found by this heuristic. Several variations have been proposed to speed up and/or improve the standard nearest neighbor algorithm (Reinelt 1994).

Insertion heuristics start with tours on small subsets (which may also include "trivial tours" on one or two nodes) and then extend these tours by inserting the remaining nodes. The choice of initial tour may be quite important. In the case of Euclidean TSPs, the convex hull[2] of the nodes provides an excellent starting tour; it was shown by Flood (1956) that the Euclidean TSP has an optimal solution that visits the cities on the boundary of the convex hull in the same order as if the boundary of the convex hull itself were traced. Therefore in many heuristics developed for Euclidean TSP, vertices on the boundary of the convex hull and the boundary itself, which forms a subtour, is used as the initial subtour (Wiorkowski and McElvain 1975; Stewart 1977; Norback and Love 1977; Norback and Love 1979). Several variations

[2]The **convex hull** of a set of V vertices is the smallest convex set that includes V.

of the criteria for the insertion scheme have also been proposed and studied (Reinelt 1994). Rosenkrantz, Stearns, and Lewis (1977) show that the *nearest insertion* (insert the node that is nearest to a tour node) and the *cheapest insertion* (among all nodes not covered so far, choose a node whose insertion causes lowest increase in the length of the tour) rules give tours that are less than twice as long as an optimal tour if the triangle inequality holds.

Heuristics based on spanning trees use a minimum spanning tree as a basis for generating tours. In problem instances where the triangle inequality is satisfied, performance guarantees of at most two times the optimal tour length are possible for these type of heuristics (Christofides 1976; Cornuéjols and Nemhauser 1978; Johnson and Papadimitriou 1985). A variant of the minimum spanning tree heuristic proposed by Christofides (1976) produces a tour which is at most 1.5 times as long as an optimal tour.

Savings heuristics were originally developed for Vehicle Routing Problems (Clarke and Wright 1964; Potvin and Rousseau 1990). This heuristic successively merges short tours to eventually obtain a Hamiltonian tour. At every iteration, the pair of subtours that give the largest savings is merged. A variation of this heuristic is a greedy type heuristic where the algorithm starts with a system of n paths of length 0, and then check, in each step, if the shortest edge not considered so far can be used to join two paths. Frieze (1979) shows that for TSP instances where the triangle inequality is satisfied, the greedy tour can be almost $(\log n)$ times as long as an optimal tour.

The second type of heuristic approaches to solving the TSP, namely *improvement heuristics*, is based on altering the current tour by certain type of basic

moves. A simple modification of the tour can be achieved, for example, by either *node insertion* or *edge insertion*. The main idea is to consider changing the position of a given node or an edge in the tour. At each iteration, the best insertion is selected until no improvement is possible (Reinelt 1994).

Another type of tour modification procedure is to exchange r edges in a feasible tour for r edges not in that solution, as long as the result remains a tour and the length of that tour is less than the length of the previous tour. These type of edge exchange procedures are called the *r-opt* procedures where r is the number of edges exchanged at each iteration (Croes 1958; Lin 1965; Lin and Kernighan 1973). All exchanges of r edges are tested until there is no feasible exchange that improves the current solution. Of course, in general, the larger the value of r, the more likely it is that the final solution is optimal. However, since the number of operations necessary to test all r exchanges increases rapidly as the number of cities increase, the values $r = 2$ and $r = 3$, are the ones most commonly used.

Lin and Kernighan (1973) proposed a variable r-opt algorithm, in which the number of edges to exchange is decided at each iteration. The empirical performance of this algorithm seems to be quite successful. A more detailed discussion of improvement heuristics can be found in (Golden and Stewart 1985; Johnson 1990; Bentley 1992; Reinelt 1994).

A third heuristic approach to solve the TSP is to use a combination of construction and improvement heuristics where the solutions obtained from construction heuristics are used as starting tours for the improvement type heuristics.

Improvement heuristics discussed here are essentially local search heuristics, where the solution found is locally optimal. That is, although the solution

is not optimal, no further improving moves can be generated. One way to obtain better solutions is to start improvement heuristics from many different starting tours to increase the chance of finding better local optima. Another possibility is to allow moves that increase the length of the tour, to escape "bad" local optima, and to restart the heuristic. Approaches using Simulated Annealing, Tabu Search, Genetic Algorithms, or Neural Networks are those kinds of heuristics which allow random "jumps" to a worse solution with respect to some criteria. For more information on Simulated Annealing and its applications to TSPs the reader is referred to (Aarst and Korst 1989a; Aarst and Korst 1989b; Johnson, Aragon, Mcgeoch, and Schevon 1991; Collins, Eglese, and Golden 1988; Kirkpatrick 1984; Cerny 1985; Johnson 1990). For Tabu Search see (Glover 1989; Knox and Glover 1989; Malek, Guruswamy, Owens, and Pandya 1989). For Genetic Algorithms see (Goldberg 1989; Mühlenbein, Gorges-Schleuter, and Krämer 1988; Ulder, Pesch, van Laarhoven, Bandelt, and Aarts 1990). Finally more information on Neural Networks and its applications to TSP can be found in (Durbin and Willshaw 1987; Fritzke and Wilke 1991; Potvin 1993).

When solving hard combinatorial problems, in addition to finding good feasible solutions, one would like to have some guarantee on the quality of the solutions found. Lower bounds (for minimization problems) on the optimal objective function value provide such guarantees. In general, lower bounds are obtained by solving a relaxation of the original problem. Since relaxations lift some of the constraints of the original problem, their sets of feasible solutions include the feasible solution set of the original problem as a (proper) subset. Thus the relaxed problem gives a lower bound for the objective function value of the optimal solution of the original (minimization) problem. Different relaxations give different lower bounds. In general, the main goal

is to find relaxations of the original problem which are efficiently solvable and which are as tight as possible. Those lower bounds then provide practitioners an impression of the performance of a heuristic on a particular problem. Lower bounds are also an important ingredient in branch-and-bound (B&B) type solution algorithms which aim to find the exact optimal solution. Lower bounds for the TSP are found from linear programming relaxations, "combinatorial" relaxations which are derived directly as obvious relaxations of the definition of a tour, or from Lagrangian relaxation methods.

The first and most straightforward relaxation of TSP is the *assignment problem* with TSP objective or Lagrangian objective function, which is obtained by relaxing the subtour elimination constraints (Little, Murty, Sweeney, and Karel 1963; Murty 1968; Bellmore and Malone 1971; Smith, Srinivasan, and Thompson 1977; Carpaneto and Toth 1980).

Another relaxation is the *1-tree problem* which is based on the observation that a Hamiltonian tour consists of a special spanning tree (namely a path) on $(n-1)$ nodes of the network plus two edges connecting this spanning tree to the last node (Held and Karp 1970; Held and Karp 1971; Christofides 1970; Helbig-Hansen and Krarup 1974; Smith and Thompson 1977; Volgenant and Jonker 1982; Gavish and Srikanth 1986; Mirchandani and Francis 1990; Fischetti and Toth 1993).

A third type of relaxation for TSP is the *2-matching problem* where the objective is to find a set of edges such that every node of the network is incident to exactly two of those edges. *n-path relaxation* of the TSP is obtained by a path of length n (n edges) that starts and ends at a given node with possible repetitions of the nodes (Houck, Picard, Queyranne, and Vemuganti 1980; Christofides, Mingozzi, and Toth 1981). The *subtour elimination relaxation* is obtained by forbidding short

cycles in the 2-matching relaxation (Padberg and Grötschel 1985; Grötschel, Lovász, and Schrijver 1988; Boyd and Pulleyblank 1990).

The lower bounds obtained from the relaxations given above are used in B&B algorithms to solve TSP optimally (Balas and Toth 1985; Miller and Pekny 1991; Miller, Pekny, and Thompson 1991). A second type of exact solution procedure called the branch-and-cut (B&C) method is one of the most effective methods for exact solutions up to date (Reinelt 1994). In those algorithms, linear programming lower bounds are obtained by optimizing the objective function over a polytope[3] P such that $P_T \subseteq P$ (where P_T is the traveling salesman polytope). To get tighter bounds, a deeper knowledge of the polytope P_T is required. Unfortunately, finding the complete description of P_T with linear equations and inequalities is almost impossible, because the number of those equations and inequalities is too large to be listed explicitly (Grötschel and Padberg 1985; Naddef and Rinaldi 1991; Naddef and Rinaldi 1993; Boyd and Cunningham 1991; Christof, Jünger, and Reinelt 1991). One way to overcome this difficulty is to generate inequalities "as needed" using the *cutting plane method.* Since the entire description of the traveling salesman polytope cannot be known, it is not possible to identify all violated inequalities describing P_T. Therefore, the cutting-plane method provides a lower bound for the TSP. The next step is to follow a B&B type approach. This procedure is called the *Branch-and-Cut* approach (Crowder and Padberg 1980; Padberg and Rinaldi 1991; Grötschel and Holland 1991).

As discussed earlier, one of the most important characteristics of the TSP is that it has a lot of variations and generalizations. One of the most well-known is the *Vehicle Routing Problem* (VRP). In VRPs there is a fleet of vehicles at the

[3]a **polytope** is a bounded polyhedron (Nemhauser and Wolsey 1988).

starting point which may be the base location of the vehicles, and the objective is to determine for each vehicle which customers should be served and in what order. Vehicles may have capacity constraints or there may be time windows for customers indicating the interval of time in which they can receive service. We will not discuss many variations and generalizations of the TSP in this thesis. However, we will introduce one generalization, namely the *Time-Dependent Traveling Salesman Problem* (TDTSP), and study a special case of this problem, the *Delivery Man Problem* (DMP) in more detail. This is because a variation of the DMP appears as a subproblem of the problem we study in this thesis.

1.3.2 The Time-dependent TSP

Consider a variation of TSP where there are n time periods, indexed by k, where each time period represents the travel between two cities on the tour. Let c_{ijk} be the cost of going directly from city i to city j in period k. Let the binary variables x_{ijk} define whether the travel from city i to city j is made at time period k. Thus the problem can be written (Fox, Gavish, and Graves 1980) as:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{j=k}^{n} c_{ijk}x_{ijk} \qquad (1.3.2.a)$$

$$\text{s.t.} \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad i = 1,..,n \quad (1.3.2.b)$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad j = 1,..,n \quad (1.3.2.c)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \qquad k = 1,..,n \quad (1.3.2.d)$$

$$\sum_{j=1}^{n}\sum_{k=2}^{n} kx_{ijk} - \sum_{j=1}^{n}\sum_{k=1}^{n} kx_{jik} = 1 \qquad i = 1,...,n \quad (1.3.2.e)$$

where for notational simplicity, it is assumed that the tour begins and ends in city

1. Constraints (1.3.2.e) ensure that for all cities i, $i = 1, ..., n$, the entering time in a given city is one less than the leaving time. Clearly, this formulation contains n^3 variables and $4n - 1$ constraints. Fox, Gavish, and Graves (1980) also give a more compact formulation with only n variables by replacing constraints (1.3.2.b, 1.3.2.c, 1.3.2.d) by

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{j=k}^{n} x_{ijk} = 1.$$

Earlier studies and formulations of the problem were given by Miller, Tucker, and Zemlin (1960), Hadley (1964), and Picard and Queyranne (1978). Recently, Gouveia and Voß (1995) presented a classification of formulations for TDTSP and Vander Wiel and Sahinidis (1996) introduced an exact solution approach.

Observe that if the cost c_{ijk} depends only on the distance, d_{ij} between two cities, then the problem reduces to the standard TSP. Thus the formulation for TDTSP provides a more compact formulation for TSP as well, at the expense of having n^3 variables instead of n^2 variables.

1.3.3 The Delivery Man Problem (DMP)

In the formulation of the TDTSP given in the previous section, consider the case where $c_{ijk} = (n-k+1)d_{ij}$. Now we have another special case of the TDTSP which is called the Delivery Man Problem. DMP minimizes the sum of arrival times at each city (or, equivalently, average arrival time at each city) in a path through all cities (Minieka 1989; Lucena 1990; Simchi-Levi and Berman 1991; Fischetti, Laporte, and Martello 1993). This problem has also been called the *TSP with cumulative costs* (Fischetti, Laporte, and Martello 1993; Bianco, Mingozzi, and Ricciardelli 1993), the *Traveling Repairman Problem* (TRP) (Afrati, Cosmadakis, Papadimitriou, Papageorgiou, and

Papakonstantinou 1986; Tsitsiklis 1992), as well as the Minimum Latency Problem (MLP) (Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan, and Sudan 1994; Goemans and Kleinberg 1996).

Before formulating the problem, let $G = (V, E)$ be an undirected network with vertex set $V = \{v_1, ..., v_n\}$ and edge set $E = \{(v_i, v_j) : v_i, v_j \in V\}$. For any $x, y \in G$, let $d(x, y)$ denote the length of a shortest path between x and y. Suppose that a server located at v_0 must visit each vertex of G and return to v_0. If the server visits the vertices in sequence $v_1, ..., v_n$ where $v_n = v_0$, then the total distance traveled by the server (or total length of the tour $(v_1, ...v_n)$) is

$$d(v_1, ..., v_n) = d(v_0, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1}). \tag{1.1}$$

The problem of finding the tour which minimizes the total traveling distance is the well-known Traveling Salesman Problem (TSP) (see Section 1.3.1)

If we assume that the traveling time is directly proportional to the actual distance traveled, then the TSP is to find the tour which visits all the vertices in the least time. The *Delivery Man Problem* (DMP) is a variant of the TSP in which we want to minimize the sum of the arrival times of the server at each vertex. In other words we seek a tour that minimizes the total waiting time of all the vertices. Afrati, Cosmadakis, Papadimitriou, Papageorgiou, and Papakonstantinou (1986) study a variant of the problem and solve the problem on a line in polynomial time by a dynamic programming procedure. Minieka (1989) shows that a simple route, in which the nodes of the tree are visited according to the depth-first rule starting from the root, is an optimal tour for the DMP on a tree with unit edge weights. He also proposes an exponential time algorithm for the problem on tree networks. Bianco,

Mingozzi, and Ricciardelli (1993) and Lucena (1990) give exponential algorithms to solve the problem on general metric spaces. Recently, Goemans and Kleinberg (1996) gave algorithms which provide an approximation ratio of 21.55 for general metric spaces and 3.5912 for trees. However, no known polynomial time algorithm yet exists to solve the problem even on tree networks.

If the vertices are visited in the order $v_1, ..., v_n$, starting from v_0, where $v_n = v_0$, then the time the server arrives at vertex v_j, denoted by $T(v_j)$, is

$$T(v_j) = \sum_{i=0}^{j-1} d(v_i, v_{i+1}). \tag{1.2}$$

Now, the Delivery Man Problem is to find a tour that minimizes the sum of the arrival times, i.e.

$$Min \quad \sum_{j=1}^{n} T(v_j). \tag{1.3}$$

Since the TSP is concerned with minimizing only the last component of (1.3), one would expect that the delivery man problem is at least as difficult as the TSP. Indeed, (Sahni and Gonzalez 1976) show that the DMP is NP-Hard.

Let us rewrite each arrival time in its open form, i.e.

$$\begin{aligned} T(v_1) &= d(v_0, v_1) \\ T(v_2) &= d(v_0, v_1) + d(v_1, v_2) \\ &\vdots \\ T(v_n) &= d(v_0, v_1) + d(v_1, v_2) + \cdots + d(v_{n-1}, v_n). \end{aligned}$$

The summation of all the arrival times then gives

$$\sum_{j=1}^{n} T(v_j) = \sum_{i=0}^{n-1} (n-i) d(v_i, v_{i+1}). \tag{1.4}$$

Now, let us consider another variant of the DMP. Consider a case where each vertex v_i, $i = 1, ..., n$, has a nonnegative weight w_i associated with it. This may depend, for example, on the urgency of the job on a machine which the repairman has to fix, or it may correspond to the amount of demand at customer points; thus we may want to make sure that customers with more demands will receive service earlier. That means we want to minimize the sum of weighted arrival times in the system. Thus the problem (1.3) becomes

$$Min \quad \sum_{j=1}^{n} w_j T(v_j). \tag{1.5}$$

This variant of the DMP appears as a subproblem of the problem we study in this thesis, which will be discussed further in Section 1.4. To our knowledge, this variant has not yet appeared in the literature. Next, we introduce the problem more formally. Since this problem is a special case of the problem we study and the findings are not directly related to the main topic of this thesis, we leave discussions on a new approach to solving it on tree networks to Appendix A. We develop a lower bound for the problem on tree networks and propose a branch-and-bound type algorithm to solve it.

We call this problem the *Weighted Delivery Man Problem* (WDMP). Observe that one can in fact reduce the WDMP to the ordinary DMP by augmenting the network in such a way that the new network has w_i nodes for each original node v_i that are connected to each other with edges of length 0. Then solving the original DMP over $W = \sum_{i=1}^{n} w_i$ nodes solves WDMP. This approach, of course, assumes that the weights are either integers or rational numbers. Aside from that, it is also possible that the problem size may increase very rapidly even for small problems when

the variation of weights associated with demand nodes is relatively large. Therefore, it is still desirable to study solution approaches to the WDMP utilizing its original formulation.

Following the steps of the previous analysis, and rewriting the arrival times in their open form, the problem (1.5) becomes

$$Min \quad \sum_{j=0}^{n-1} \left(\sum_{i=j+1}^{n} w_i \right) d(v_j, v_{j+1}). \tag{1.6}$$

Let $W = \sum_{i=1}^{n} w_i$ denote the total weight in the system. Then the problem becomes

$$Min \quad \sum_{j=0}^{n-1} \left(W - \sum_{i=1}^{j} w_i \right) d(v_j, v_{j+1}). \tag{1.7}$$

In the ordinary DMP or the TDTSP, the cost of travel on an edge depends on the position of that edge in the given tour. However in the case of the WDMP the cost of travel on a given edge not only depends on its position but also on the set of vertices that are visited *before* the first travel began on the given edge (or equivalently on the set of vertices that are visited *after* the first travel on the given edge).

It is straightforward to show that when all weights are equal the problem reduces to the ordinary DMP defined earlier. Thus the WDMP is at least as difficult as the DMP. That is the WDMP is also NP-Hard.

1.4 Location-Routing Problems and The Supply Connected Location-Allocation Problem (SCLAP)

Up to now, we have discussed the problems in the literature which we believe are directly related to the problem we consider in this thesis. In this section we will

first provide a brief discussion on Location-Routing Problems. This is because our problem can be put in this category. Later we will introduce the problem we study and explain its relation to the problems we have discussed so far.

There exist several practical situations where the need arises to locate one or several facilities, allocate demand to be served to those facilities, and to construct associated delivery routes to multiple users. In such contexts, location, allocation and routing are intertwined decisions which must be modeled and optimized simultaneously. These problems are called *Location-Routing Problems* (LRPs) in the literature. A considerable number of recent studies in location theory deals with several versions of LRPs. Laporte (1988) provides an extensive review of this class of problems with examples and applications found in the literature.

We should also note, however, that the problem we study here does not conform to the typical forms of location-routing problems found in the literature. The main difference comes from the routing cost. Typically, in most LRPs, the routing decision is represented by a *Traveling Salesman Problem* (see Section 1.3.1) type objective where the total length of the delivery route is minimized. In our case, as will be described in the following pages, the cost of the delivery route depends not on the total length of the tour but the sum of all distances along the route from the supply plant which is the starting point of the route as well as the sum of weights of demand points that are allocated to each facility. Thus, the routing decision in our problem is represented by a *Weighted Delivery Man Problem* type objective, which, to our knowledge, is itself a new model that has not been studied in the literature.

As discussed in (Laporte 1988), many distribution systems can be represented by several layers. Many systems have three layers where the layers are

identified as *primary facilities, secondary facilities* and *users.* The primary facilities may represent factories, secondary facilities correspond to warehouses or depots, and the users represent customers. Usually, primary facilities and users are located at known and fixed locations and the number or location of the secondary facilities, together with the associated distribution routes constitute decision variables. Of course there may be other distribution systems which consist of two layers or more than three layers. For example, in the p-median problem (see Section 1.2.1) the first layer consists of facilities to be located and the second layer consists of demand points.

The problem in this study is a three layer distribution system. There is one primary facility at a known location which we term as the "supply plant", and n "demand points" as users. p "facilities" are the secondary facilities to be located at p of m predetermined sites ($p \leq m$). Note that the predetermined facility sites can be a proper subset of the demand point locations. Throughout this study we take all the demand point locations as candidate sites for facilities, i.e. $m = n$. Facilities receive their supply from the supply plant and distribute to demand points. The distribution modes from facilities to demand points are direct service-and-return trips and facilities get their supply via a single round trip starting from the supply plant.

The problem which we call the *Supply Connected Location-Allocation Problem* (SCLAP) involves the location of p facilities with respect to n demand points and a supply plant. The locations of the demand points and the supply plant are known and fixed. There is a positive demand associated with each demand point, and the objective is to provide service to demand points via facilities serving as transshipment or distribution points. The supply plant originates all material and

ships it in bulk to facilities along a route to be determined. We call this route the "supply route". We assume that the shipment vehicle does not need to return to the supply plant. This assumption may indeed be valid in many business applications where the carrying of the shipment is done by an outside contractor. Thus the shipment vehicle does not return to the supply plant, but rather to its base point at some other location. In cases where there is a cost for an empty vehicle to return to the supply plant, we simply add this cost to the objective function and construct a closed tour instead of a path. In the following research, we assume that the supply route consists of a single path originating at the supply plant and passing through all the facilities. After the facilities receive their shipment from the supply plant, they then distribute the material to demand points by direct service-and-return trips. Since there are no capacity constraints on facilities, each demand point receives all of its shipment from the facility which provides the lowest transportation cost. Any ties can be broken arbitrarily.

The cost of transportation for a given demand point consists of two parts. The first part is the cost of receiving the shipment from its designated facility. By designated facility we mean the one to which that demand point is allocated. This portion of the cost is directly proportional to the weight (amount of demand) of the given demand point and the distance between that demand point and its designated facility. The second part of the cost is the cost of sending that demand point's shipment from the supply plant to its designated facility on the supply route. The cost of transporting the demand to be distributed by a facility from the supply plant, is proportional to the total amount of demand allocated to this facility and the total traveling distance to reach this facility on the supply route. The total traveling

distance to reach the given facility is determined by the sum of the distances traveled between consecutive facilities on the supply route, starting from the supply plant up to this facility. The shipments from the supply plant to facilities on the supply route are done in bulk, that is all the material requirements for all the demand points that are allocated to a facility are sent together in bulk to that facility. In practice, bulk shipments are generally less costly. In accordance with this observation, in our problem the cost of transportation on the supply route is discounted by multiplying the cost by a factor α $(0 < \alpha < 1)$. Therefore the cost of shipping one unit of material per distance on the supply route is $100(1-\alpha)$ percent cheaper than the cost of shipping the same amount of material per distance elsewhere on the network. We call α the "discounting factor".

The following example illustrates the problem. Suppose that a petroleum plant is operated at some distant location. The product is shipped in bulk to the facilities along a route connecting them; it is then distributed to demand points that are allocated to the facilities. Examples of bulk shipment could involve rail, containers or a pipeline. Delivery from facilities could be in smaller volume (by truck) and thus be more expensive.

Observe that when the cost of transportation on the supply route is zero $(\alpha = 0)$, the problem reduces to finding locations of p facilities that minimizes the sum of weighted distances from each demand point to a nearest facility. This problem is the well-known p-median problem which we studied in Section 1.2.1.

Now, consider the case when the transportation cost on the supply route is not discounted (i.e $\alpha \geq 1$). Then all the facilities would be located at the supply plant and the demand points would receive their shipments directly from the supply

plant. This is because of the triangle inequality property of distance functions on any metric space, including networks (when the distance between any two points in the network is defined to be the length of a shortest path between them). The length of the direct path (or the shortest path) from the supply plant to a given demand point is always less than or equal to the length of any other path passing through one or more intermediate points (facilities).

For another special case of the problem, consider that the locations of p facilities and the allocations of demand points to facilities are somehow known. We still have to determine a least cost route visiting all facilities at least once.

The total travel distance from the supply plant to a given facility depends on the sequence of facilities visited before that facility. Let t_{0k} be the total traveling distance from the supply plant to reach the facility which is positioned at k, i.e. traveling distance to reach that facility after $k-1$ facilities are served before it. Let $j(k)$ denote the location of the facility which is visited at position k on the supply route and let $d_{j(k),j(k+1)}$ denote the distance between any two facilities that are visited by the service truck at consecutive positions k and $k+1$. Then we have

$$\begin{aligned} t_{01} &= d_{0,j(1)} \\ t_{02} &= d_{0,j(1)} + d_{j(1),j(2)} \\ t_{03} &= d_{0,j(1)} + d_{j(1),j(2)} + d_{j(2),j(3)} \\ &\vdots \\ t_{0n} &= d_{0,j(1)} + d_{j(1),j(2)} + \cdots + d_{j(p-1),j(p)} \end{aligned}$$

Let $\bar{w}_{j(k)}$ be the total amount of demand that is allocated to facility at site j which

is positioned at k in the supply route. Then the total cost of transportation from the supply plant to all facilities will be

$$\sum_{k=1}^{p} \alpha \bar{w}_{j(k)} t_{0k} = \sum_{k=1}^{p} \sum_{l=k}^{p} \alpha \bar{w}_{j(l)} d_{j(k-1),j(k)}. \tag{1.8}$$

The RHS of (1.8) gives an equivalent formulation of the transportation cost on the supply route. That is the sum of the transportation costs between consecutive facilities on the supply route (for now, consider the supply plant as a facility too). The cost of transportation between consecutive facilities is proportional to the distance between the facility pair, the demand in the system yet to be served, and a scale factor α.

The problem of minimizing (1.8) over all possible routes visiting all facilities at least once gives the Weighted Delivery Man Problem (see Section 1.3.3) when we assume that we can return to the origin from any point without any cost, thus having a complete tour instead of a path.

For another variant of the SCLAP, consider the case where the number of facilities to locate is not known. Additionally, instead of including facility location costs implicitly by constraining the number of facilities (see discussions in Section 1.2.2) in the model, there is now a cost of building the facility at each candidate site. In this case, we want to open a certain number of facilities and assign demand points to them, as well as construct a delivery route for supply from the supply plant to facilities, with the least possible building and transportation cost. It is a straightforward observation that, in the case when the cost of supplying facilities is ignored, the problem reduces to the uncapacitated facility location problem discussed in Section 1.2.2.

In this thesis, we formulate and develop solution approaches for a problem which is a combination of two different class of problems that are considered in the literature, namely the location-allocation and routing problems. By doing so, we hope to be able to provide a basis for some of those problems that appear in practice which require the location, allocation and routing decisions to be optimized simultaneously. There are, of course, certain limitations to our model. First, we developed the model for a deterministic problem. That is, we assume that the demands for each demand point is known and fixed. However, in practice, demands are often stochastic in nature, which means that their exact values are not known. In those cases, objectives become minimization of the expected value of the total transportation cost. Another limitation in our model is the assumption that the facilities have unlimited capacity. In some types of problems that assumption may be valid: for example, when the facilities are warehouses where relatively small sized items, e.g. most postage mail items, are stored for a short period of time before being rerouted to their final destinations and the cost of building the warehouses does not increase significantly with the increase in volume of material to be handled there. However, there are many other cases where the assumption of unlimited capacity for facilities cannot be valid, either due to financial or physical limitations.

Third, we assume, in our model, that the companies market share and the demand points it will serve are already determined and will not change due to presence of a competitor and its actions. Examples in which this assumption holds can be found in many public service operations, where the facilities together serve an entire community. However, especially in private sector, we can find many other examples in which the assumption may no longer be valid. This is because the market

share and the operations of the company can be greatly effected by the actions of its competitors in the same market. Approaches to relax those limitations of the model can provide a basis for development of new models and variants of our problem.

Since the two special cases of SCLAP, namely the p-median and DMP are $\mathcal{NP}$-Hard on general networks, the problem is quite difficult in general. In this study we use two approaches. The first is to identify some solvable special cases of the problem on simple networks, such as the line. The second is to provide an exact algorithm to solve the problem together with some heuristic algorithms and test those algorithms for performance in terms of closeness to optimality and run time.

In Chapter 2 we study SCLAP on a line and a tree and give efficient solution procedures using dynamic programming (DP). The case on a tree is a restricted version where we assume that facilities will be located on a single path to avoid traveling the same distance more than once. In Chapter 3, we study the problem on general networks and present a single assignment branch-and-bound algorithm to solve it. In Chapter 4, we develop four local search type heuristics and provide computational results. We compare the heuristics in terms of closeness to optimality (whenever the optimum solutions are available) or to the best solution found and run time. Finally, Chapter 5 gives a brief summary of the thesis and concludes with remarks on further research.

Chapter 2

SCLAP on a Line and a Tree

In this chapter, we study the problem on a line and on a special case of a tree network. A polynomial time backward dynamic programming solution procedure is provided for the case on a line. The problem on tree networks is for the case where the facilities are to be located on a single path. This problem, after certain steps, then reduces to the case on a line, for which we already have a polynomial time solution algorithm. The general form of the problem on tree networks is still an open question.

2.1 The Case on a Line

Assume that demand points are located on a line segment $[0, \ell]$ and that the main supply facility is located at point $a_0 = 0$. Let a_i, $i = 1, ..., n$, be the locations of demand points. For ease of notation and further convenience we assume that demand point locations are indexed such that $a_0 = 0 < a_1 < a_2 < \cdots < a_n = \ell$. Let $w_i \geq 0$, $i = 1, ... n$, denote the amount of demand at corresponding demand point i. We want to locate p facilities and allocate demands to them such that the overall cost of serving all the demand points as well as the facilities is minimized. Let $X = \{x_1, ..., x_p\}$ be the set of facility locations. We now formulate the problem as follows:

$$\text{Minimize} \quad \sum_{j=1}^{p}\sum_{i=1}^{n} w_{ji}|x_j - a_i| + \alpha \sum_{j=1}^{p-1}\sum_{k=j+1}^{p}\sum_{i=1}^{n} w_{ki}|x_j - x_{j+1}| + \alpha W|x_1 - a_0| \quad (2.1.a)$$

$$\text{subject to} \quad \sum_{j=1}^{p} w_{ji} = w_i \qquad i = 1, ..., n \quad (2.1.b)$$

$$w_{ji} \geq 0 \qquad j = 1, ..., p;\ i = 1, ..., n, \quad (2.1.c)$$

where

w_{ji}: proportion of demand at demand point i served by the facility j

w_i: total amount of demand at demand point i

$W = \sum_{i=1}^{n} w_i$: total demand in the system.

Before we introduce a solution procedure for this problem we should state some of its fundamental properties.

Assume that all the locations of facilities and demand allocations are known but that the sequence (or indexing) of the facilities is unknown. In this case we have p known locations and we also know the amount of demand allocated to these points. The problem then reduces to finding the best routing to satisfy demands allocated at these points. The starting point is a_0, the original supply plant location. Although this problem is difficult to solve on general networks, it turns out to be easily solvable on a line when the supply plant is located at one end of the line. The fact that all the demand in the system that is not satisfied yet is transported to the next facility forces the facilities to be visited according to their positions on the line from the original supply plant. This property is stated in the next observation.

Observation 2.1 *In the optimal solution, facilities will be located on* $[0, \ell]$ *such that* $0 \leq x_1 \leq x_2 \leq \cdots \leq x_p \leq \ell$.

The next lemma suggests that a demand point will be allocated to one of

the facilities located nearest to it on each side. Note, however, that the one finally selected may not be the nearest of the two.

Lemma 2.1 *In the optimal solution, the demand at demand points will be fully allocated to one of the two facilities that are adjacent from left and right.*

Proof: Assume that the locations $x_1, ..., x_p$ of facilities are given and we want to allocate the demand optimally. According to Observation 2.1 we can consider only the cases where $0 \leq x_1 \leq x_2 \leq \cdots \leq x_p \leq \ell$. Let a' be the location of a given demand point.

Since there are no capacity constraints, the demand at a demand point will be fully allocated to the one facility which provides the lowest cost, unless a tie occurs. In the case of a tie, however, we can still assume without loss of generality that in one of the optimal solutions, each demand point will be fully allocated to one facility only.

Now, let us allocate one unit of demand at a' to one of the new facilities with the least transportation cost. Assume that the demand is allocated to facility x_j. The unit cost of transportation will then be:

$$\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j-1} |x_k - x_{k+1}| + |x_j - a'|, \tag{2.1}$$

i.e. the scaled transportation cost from the supply plant up to the j-th facility via previous transshipment points (lower indexed facilities) plus the cost of transportation from j-th new facility to the demand point at a'.

Since we have $x_1 \leq x_2 \leq \cdots \leq x_p$, the two nearest facilities to a' from left and right will be a consecutively indexed pair of facilities. Let j and $j+1$ be the

indexes of new facilities that are nearest to a' from left and right. Consider a lower indexed, say $j-1$, facility than the j-th facility. Since the location space is a line and facility locations are such that $x_1 \leq x_2 \leq \cdots \leq x_p$ we have:

$$|x_{j-1} - x_j| + |x_j - a'| = |x_{j-1} - a'|. \tag{2.2}$$

Then the unit cost of allocating the demand at a' to x_{j-1} will be

$$\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j-2} |x_k - x_{k+1}| + |x_{j-1} - a'|, \tag{2.3}$$

and the unit cost of allocating demand at a' to x_j will be

$$\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j-2} |x_k - x_{k+1}| + \alpha|x_{j-1} - x_j| + |x_j - a'|. \tag{2.4}$$

Since $\alpha \in [0, 1]$,

$$\alpha|x_{j-1} - x_j| + |x_j - a'| \leq |x_{j-1} - x_j| + |x_j - a'|, \tag{2.5}$$

and (2.2) gives $|x_{j-1} - a'|$ for RHS. The unit cost of transportation will be less when the demand at a' is allocated to facility at x_j. Hence, we need not consider any further away (lower indexed) facilities on the left.

Now consider a higher indexed, say $j+2$, facility than $(j+1)$st. By similar arguments as above we have

$$|x_{j+1} - x_{j+2}| + |x_{j+1} - a'| = |x_{j+2} - a'|. \tag{2.6}$$

The unit transportation cost for facility $j+1$ is:

$$\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j} |x_k - x_{k+1}| + |x_{j+1} - a'|, \tag{2.7}$$

and unit transportation cost for facility $j+2$ is:

$$\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j} |x_k - x_{k+1}| + \alpha|x_{j+1} - x_{j+2}| + |x_{j+2} - a'|. \tag{2.8}$$

Clearly from (2.6) and $\alpha \in [0,1]$ the transportation cost for facility $j+1$ is less than or equal to that of a higher indexed facility. So we need not consider further away facilities on the right of a' as well. This means that all the demand at a' will be allocated to either of the two nearest facilities on the left and right of a'. □

Now we will explore to which one of the facilities nearest on the left and right of a' to allocate the demand. From (2.4) and (2.7) we can see that the cost of transshipment up to the j-th facility is the same for both facilities j and $j+1$, which is $\alpha|x_1 - a_0| + \alpha \sum_{k=1}^{j-1} |x_k - x_{k+1}|$. Let A denote this cost; then the costs of transportation for facilities j and $j+1$ will be

$$A + |x_j - a'| \tag{2.9}$$

$$A + \alpha|x_j - x_{j+1}| + |x_{j+1} - a'|, \tag{2.10}$$

respectively.

Clearly, the relationship between (2.9) and (2.10) will determine the facility to which a' will be allocated, i.e. a' will be allocated to the facility which gives a lower transportation cost. By comparing costs (2.9) and (2.10) and rearranging the inequalities, we can easily establish the following relationship.

Proposition 2.1 *Let a' be a demand point location between facility location x_j and x_{j+1}, then*

(i) *if $a' < \frac{1}{2}[(1-\alpha)x_j + (1+\alpha)x_{j+1}]$ then a' is allocated to facility j*

(ii) *if $a' > \frac{1}{2}[(1-\alpha)x_j + (1+\alpha)x_{j+1}]$ then a' is allocated to facility $j+1$*

(iii) *if* $a' = \frac{1}{2}[(1-\alpha)x_j + (1+\alpha)x_{j+1}]$ *then* a' *can be allocated to either facility* j *or facility* $j+1$.

As a final step before introducing a dynamic programming solution procedure for SCLAP we need to establish the following lemma which is essentially the same as the *fundamental insight* in (Love 1976; Love, Morris, and Wesolowsky 1988).

Lemma 2.2 *(Fundamental Insight) Assuming that the demand points are indexed such that* $a_0 = 0 \leq a_1 \leq a_2 \leq \cdots \leq a_n = \ell$. *they will be optimally allocated to facilities in sequence, i.e. if demand points 1 and 4 are allocated to a given facility so will the demand points 2 and 3.*

Proof: Assume that the locations of facilities are given. Let v be a demand point location between facility locations x_j and x_{j+1}. We should prove that if v is allocated to facility j then all the demand points on the path between x_j and v will be allocated to x_j as well: and similarly if v is allocated to facility $j+1$, then all the demand points on the path between x_j and v will be allocated to x_{j+1} as well.

Case 1: v is allocated to x_j. Let p be a demand point located on the path between x_j and v. Then we have:

$$|x_j - v| \leq \alpha|x_j - x_{j+1}| + |x_{j+1} - v| \tag{2.11}$$

$$|x_j - p| + |p - v| \leq \alpha|x_j - x_{j+1}| + |x_{j+1} - v| \tag{2.12}$$

$$|x_j - p| + |p - v| \leq \alpha|x_j - x_{j+1}| + |x_{j+1} - v| + |p - v| \tag{2.13}$$

$$|x_j - p| \leq \alpha|x_j - x_{j+1}| + |x_{j+1} - p|. \tag{2.14}$$

(2.11) comes directly from (i) and (iii) of Proposition 2.1, i.e. when (2.9) is less than or equal to (2.10). Since the location space is a line and p is located between x_j and v, we have $|x_j - p| + |p - v| = |x_j - v|$ which justifies (2.12). Adding $|p - v| \geq 0$ to RHS of (2.12) gives (2.13). Finally dropping $|p - v|$ from LHS of the inequality (2.13) and substituting $|x_{j+1} - p| = |x_j - v| + |p - v|$ on the RHS gives (2.14) which suggests that p is also allocated to x_j.

Case 2: v is allocated to x_{j+1}. Let q be an existing facility location between v and x_{j+1}. With similar arguments as above it is straightforward to show that q will also be allocated to x_{j+1}.

Cases 1 and 2 complete the proof. □

2.1.1 Dynamic Programming Solution Procedure

In this section, we introduce a backward dynamic programming solution procedure to SCLAP on a line which is similar in principle to the one given in (Love 1976) for the standard location-allocation problem except that we use a backward procedure instead of a forward one. It is also possible to set up the DP procedure presented here as a forward procedure.

Let us first resolve some important considerations. Assume that we want to locate the j-th facility optimally. Let demand points $a_s, a_{s+1}, ..., a_t$ be allocated to it. The problem then reduces to

$$\text{Minimize } \sum_{i=s}^{t} w_i|x_j - a_i| + \alpha(\sum_{i=s}^{n} w_i)|x_{j-1} - x_j| + \alpha(\sum_{i=t+1}^{n} w_i)|x_j - x_{j+1}| \qquad (2.15)$$

Observe that if the locations x_{j-1} and x_{j+1} of facilities $j-1$ and $j+1$ are known, the problem becomes a well-known 1-median problem which can be solved very efficiently.

Finding the optimal location for the 1-median problem on a line requires no knowledge of distances because it has been shown to be the demand point location for which the sum of the weights on one side becomes greater than or equal to the sum of the weights on the other side when the weight of the given facility is added (Goldman 1971). This property and the following corollary to Lemma 2.2 makes it possible to find the optimal location in (2.15) without knowing the actual locations of x_{j-1} and x_{j+1}.

Corollary 2.1 *Let $s, s+1, ..., t$ be indexes of the demand points that are allocated to facility j. Then the locations of facilities $j-1$ and $j+1$ will be such that $x_{j-1} < a_s \leq x_j \leq a_t < x_{j+1}$.*

Once we know that the x_{j-1} and x_{j+1} will be located further away from the farthest allocated demand point to facility j on the left and right respectively, we can simply add their weights to the existing facilities a_s and a_t and find the optimal location for x_j. Note that the weights for x_{j-1} and x_{j+1} will be $(\alpha \sum_{i=s}^{n} w_i)$ and $(\alpha \sum_{i=t+1}^{n} w_i)$, respectively.

However, to be able to apply a DP procedure to SCLAP we have to find a way to set up a DP relation in terms of costs. We will use notation similar to the one given in (Love 1976). Let a given *stage* of the DP formulation be the index of the facility that is being located. The stage number is given by j, $j = p, p-1, ..., 1$. We start with locating the p-th facility first and locate the $(p-1)$st facility next, and work backward until we locate the first facility last. Let s denote the *state* defined by the index of the lowest numbered unallocated demand point. Then $j \leq s \leq n-p+j$ if $j > 1$ and $s = 1$ if $j = 1$. Let $A_j(s)$ be the set of all possibly optimal subsets

of allocations of demand points to the j-th facility, given s. Each such subset is represented by an index t_j, $s \leq t_j \leq n-p+j$ if $j \neq p$ and $t_j = n$ otherwise. Let $C^*(s,t_j)$ denote the minimum "partial" weighted distance cost of j-th facility optimally located with respect to demand points $s, s+1, ...t_j$ and facilities $j-1$ and $j+1$. Let $x_j^*(s,t_j)$ be the optimal location of facility j with demand points $s, s+1, ..., t_j$ allocated to it. The cost of transportation between facilities $j-1$, j, and $j+1$ added to the cost of shipment from facility j to demand points that are allocated to facility j, is

$$\sum_{i=s}^{t_j} w_i|x_j - a_i| + \alpha \sum_{i=s}^{n} w_i|x_{j-1} - x_j| + \alpha \sum_{i=t_j+1}^{n} w_i|x_j - x_{j+1}| \qquad (2.16)$$

Since $x^*(s,t_j)$ is defined to be the optimal location of facility j with demand points $s, s+1, ..., t_j$ allocated to it, $x^*(s,t_j)$ is also a minimizer for (2.16). Let $W(k) = \sum_{i=k}^{n} w_i$ for $1 \leq k \leq n$. We can split the cost of transportation between facilities $j-1$ and j, and also between the facilities j and $j+1$ (i.e. second and third portions of (2.16)) into two and rewrite (2.16) as follows:

$$\begin{aligned}\sum_{i=s}^{t_j} w_i|x_j - a_i| + \alpha W(s)|x_{j-1} - a_s| + \alpha W(s)|x_j - a_s| + \\ \alpha W(t_j+1)|x_j - a_{t_j+1}| + \alpha W(t_j+1)|x_{j+1} - a_{t_j+1}| = \\ \alpha W(s)|x_{j-1} - a_s| + C(s,t_j) + \alpha W(t_j+1)|x_{j+1} - a_{t_j+1}| \qquad (2.17)\end{aligned}$$

where $C(s,t_j)$ is defined to be

$$C(s,t_j) = \alpha W(s)|x_j - a_s| + \sum_{i=s}^{t_j} w_i|x_j - a_i| + \alpha W(t_j+1)|x_j - a_{t_j+1}| \qquad (2.18)$$

Then, $C^*(s,t_j)$ will be the value of $C(s,t_j)$ when (2.16) is minimized by $x_j^*(s,t_j)$.

Assume that we are at the stage j of the DP procedure, i.e. we will locate the j-th facility. Let the state be s. We have to consider all possible allocations from s on: that is, the sets of demand points $\{s\}$, $\{s, s+1\}$,...,$\{s, s+1, ..., n-p+j\}$. At this point observe that the part of the cost of transporting all the demand for facilities $j, j+1, ..., p$ from facility $j-1$ to j will be the same up to demand point s which is $\alpha W(s)|x_{j-1} - a_s|$. Only the part $\alpha W(s)|x_j - a_s|$ will change, depending on the location of x_j. So in order to find the best allocation configuration for facilities $j, j+1, ..., p$ from s on, we can minimize only the part of the cost without changing the optimal solution. Hence the DP relation will be:

$$f_j^*(s) = \min_{t_j \in A_j(s)} \{f_j(s, t_j)\} \tag{2.19}$$

where $f_j(s, t_j) = C^*(s, t_j) + f_{j+1}^*(t_j + 1)$. For the beginning stage (stage p) of the procedure we take $f_{p+1}^* = 0$.

The next example will provide an illustration of the procedure.

2.1.2 Example

In the example below, we have 9 demand points on a line of length 25 units and we want to locate 4 facilities. The scaling factor α is taken to be 0.5., i.e. $n = 9$, $p = 4$, and $\alpha = 0.5$. The locations, a_i of demand points and the amounts of demand (weight) at each demand point are given in Table 2.1. Note that the weight of the supply plant located at $a_0 = 0$ equals $\alpha \sum_{i=1}^{9} w_i$. Tables 2.2, 2.3, 2.4 and 2.5 show the stages of the DP procedure.

We find $f_1^*(1) = 175.5$ with $t_1^* = 1$, $t_2^* = 4$, $t_3^* = 6$ and $t_4^* = 9$, and an optimal solution is:

i	0	1	2	3	4	5	6	7	8	9
a_i	0	4	7	8	12	17	19	21	24	25
w_i	11	3	2	4	1	2	3	3	2	2

Table 2.1: Data for the Example Problem

s	$f_4(s, 9)$	$f_4^*(s)$	$x_4^*(s, 9)$
9	0	0	$a_9 = 25$
8	2	2	$a_8 = 24$
7	14	14	$a_7 = 21$
6	28	28	$a_6 = 19$
5	44	44	$a_6 = 19$
4	84.5	84.5	$a_6 = 19$

Table 2.2: Stage 4 of DP Algorithm, $j = 4$, $t_4^* = t_4 = 9$

	$f_3(s, t_3)$								
s/t_3	3	4	5	6	7	8	$f_3^*(s)$	t_3^*	$x_3^*(s, t_3^*)$
8						1	1	8	$a_8 = 24$
7					8	10	8	7	$a_7 = 21$
6				21	18	22	18	7	$a_6 = 19$
5			38	34	34	38	34	6,7	$a_5 = 17$
4		74	73	71.5	71.5	77.5	71.5	6,7	$a_5 = 17$
3	110.5	102	105	114.5	128	143	102	4	$a_3 = 8$

Table 2.3: Stage 3 of DP Algorithm, $j = 3$

	$f_2(s,t_2)$								
s/t_2	2	3	4	5	6	7	$f_2^*(s)$	t_2^*	$x_2^*(s,t_2^*)$
7						7	7	7	$a_7 = 21$
6					15	17	15	6	$a_6 = 19$
5				28	28	33	28	5,6	$a_5 = 17$
4			64	63	65.5	70.5	63	5	$a_4 = 12$
3		97.5	92	95	108.5	127	92	4	$a_3 = 8$
2	110.5	108	103	106.5	120	138.5	103	4	$a_2 = 7$

Table 2.4: Stage 2 of DP Algorithm, $j = 2$

	$f_1(s,t_1)$								
s/t_1	1	2	3	4	5	6	$f_1^*(1)$	t_1^*	$x_1^*(s,t_1^*)$
1	175.5	176	181	180	190	207	175.5	1	$a_1 = 4$

Table 2.5: Stage 1 of DP Algorithm, $j = 1$

$x_1^* = x_1^*(1,1) = 4, \quad w_{11}^* = w_1 = 3,$

$x_2^* = x_2^*(2,4) = 7, \quad w_{22}^* = w_2 = 2, \quad w_{23}^* = w_3 = 4, \quad w_{24}^* = w_4 = 1,$

$x_3^* = x_3^*(5,6) = 17, \quad w_{35}^* = w_5 = 2, \quad w_{36}^* = w_6 = 3,$

$x_4^* = x_4^*(7,9) = 21, \quad w_{47}^* = w_7 = 3, \quad w_{48}^* = w_8 = 2, \quad w_{49}^* = w_9 = 2,$

with all other w_{ji}'s equal to zero. That is, facility 1 is located at the location $a_1 = 4$ of demand point 1 and serves demand point 1, facility 2 is located at the location $a_2 = 7$ of demand point 2 and serves demand points 2, 3 and 4, facility 3 is located at the existing facility location $a_5 = 17$ and serves demand points 5 and 6. Finally, facility 4 is located at the demand point location $a_7 = 21$ and serves demand points 7,8 and 9.

An alternative optimum solution is:

$x_1^* = x_1^*(1,1) = 4, \quad w_{11}^* = w_1 = 3,$

$x_2^* = x_1^*(2,4) = 7, \quad w_{22}^* = w_2 = 2, \quad w_{23}^* = w_3 = 4, \quad w_{24}^* = w_4 = 1,$

$x_3^* = x_1^*(5,7) = 17, \quad w_{35}^* = w_5 = 2, \quad w_{36}^* = w_6 = 3, \quad w_{37}^* = w_7 = 3,$

$x_4^* = x_1^*(8,9) = 24, \quad w_{48}^* = w_8 = 2, \quad w_{49}^* = w_9 = 2,$

with all other w_{ji}'s equal to zero.

By similar arguments in (Love 1976) the DP procedure introduced here is $O(p(n-p)^2)$.

2.1.3 Varying α

It is quite obvious from the formulation of SCLAP given in Section 2.1 that when $\alpha = 0$ the problem reduces to a well known location-allocation problem (Love, Morris, and Wesolowsky 1988). It is also called the p-median problem on networks.

Since any unit of demand actually has to be shipped from the original plant to its point of location via some path visiting some of the facilities, when

$\alpha \geq 1$, shipment via intermediate facilities becomes unprofitable because the length of the direct path from the original supply plant to demand location is always less than or equal to the length of any other path via intermediate transshipment points (facilities). This is a direct consequence of the triangle inequality property of distance function in any metric space. Hence, unless restricted otherwise, all the p facilities will be located on top of the original supply plant when $\alpha \geq 1$. For tree networks and the line, the problem is actually equivalent to a 1-median problem. Since the major plant supplies all the demand in the system, it has a weight $\alpha \sum_{i=1}^{n} w_i$ which is greater (when $\alpha > 1$) or equal (when $\alpha = 1$) to half of the total weight in the system. By the well-known solution procedure for the 1-median problem on trees (Goldman 1971), the optimal location is the supply plant location. When $\alpha = 1$ there may be other degenerate solutions; however, without loss of generality we can still take the original plant location to be the optimal location.

Once we have found the optimal solution for SCLAP for a given value of α, the next question one may ask is to determine the range of α for which the current solution is optimal. Proposition 2.1 readily suggests a way to determine such a range. Observe that the cases (i),(ii), and (iii) of Proposition 2.1 can be written in terms of fractions such that

(i) if $\frac{2a'-(x_j+x_{j+1})}{(x_{j+1}-x_j)} < \alpha$ then a' is allocated to facility j

(ii) if $\frac{2a'-(x_j+x_{j+1})}{(x_{j+1}-x_j)} > \alpha$ then a' is allocated to facility $j+1$

(iii) if $\frac{2a'-(x_j+x_{j+1})}{(x_{j+1}-x_j)} = \alpha$ then a' can be allocated to either facility j or facility $j+1$.

Let a'' be the proportion of the distance $|x_{j+1} - x_j|$ that a' is from x_j. Then, equivalently, we obtain

(I) if $a'' < \frac{1+\alpha}{2}$ then a' is allocated to facility j

(II) if $a'' > \frac{1+\alpha}{2}$ then a' is allocated to facility $j+1$

(III) if $a'' = \frac{1+\alpha}{2}$ then a' can be allocated to either facility j or facility $j+1$.

Once the optimal solution is found, we only consider the demand points between two consecutive facilities. For each of them we calculate the value of α for which case 3 occurs. Define α_i to be the value of α that is so calculated. For demand points that are allocated to a facility on their left (cases i and iii), $1 \geq \alpha \geq \alpha_i$ is the range of α for which it will still be allocated to the same facility. For facilities that are allocated to a facility on their right (cases ii and iii) $0 \leq \alpha \leq \alpha_i$ is the range of α for which it will still be allocated to the same facility. Let α_L and α_U be the lower and upper bound on the value of α for which the current solution is still optimal, respectively. Then we have

$$\alpha \in [\alpha_L, \alpha_U] \tag{2.20}$$

where

$$\alpha_L = \max\{0, \alpha_i : a_i \text{ is allocated to a facility on its left}\} \tag{2.21}$$

$$\alpha_U = \min\{1, \alpha_i : a_i \text{ is allocated to a facility on its right}\} \tag{2.22}$$

Moreover, we can show that we do not need to consider all the demand points. Only the pairs of consecutive demand points each of which are allocated to different facilities will suffice to determine the range of α. So the $p-1$ rightmost demand points that are allocated to the first $p-1$ facilities will determine the lower bound α_l, and $p-1$ leftmost demand points that are allocated to the last $p-1$ facilities will determine the upper bound α_u. Therefore, for a given optimal solution we can determine the range of α in $O(p)$ time.

2.2 Location on a Tree

In this section we consider a slightly more general case where the location space for demand points is a tree and facilities are restricted to be located on a linear path. Let $T = (V, E)$ be a *tree* with node set $V = \{v_0, v_1, ..., v_n\}$ and edge set $E = \{(v_i, v_j) : 0 \leq i < j \leq n\}$. Let l_{ij} denote the length of arc (v_i, v_j) of T. If x_1 and x_2 are any two points on T, then there exists a unique path $P(x_1, x_2)$ joining x_1 to x_2. The distance between x_1 and x_2, denoted $d(x_1, x_2)$, is defined to be the length of $P(x_1, x_2)$. The *degree* of a node is defined to be the number of edges incident to it. A node is called a *leaf* if its degree is one.

We assume that the demand points are located on the nodes of the tree and the supply plant is located at a leaf node numbered 0 for notational convenience. If the supply node 0 is not numbered 0 originally, we can renumber the nodes of the tree such that the supply node has the number 0.

Next, we constrain all the facilities to be located on one path. Indeed, this is not an unrealistic restriction. Since during bulk shipment all the demand in the system is carried along the way until it is brought to the respective facility, it is reasonable to prefer that none of the demand should travel the same distance twice, i.e. only one way bulk shipment is allowed on any path. Restricting all facilities to be located on a linear path will ensure one way bulk shipment.

Since all the demand originates from the supply plant at node v_0, the new facilities will be located on some path $P(v_0, x)$, $x \in T$. Let $P^*(v_0, x^*)$ be the optimal path. It is straightforward to adopt Observation 2.1 to this case, which suggests that facilities are located on $P^*(v_0, x^*)$ such that $d(v_0, x_1) \leq d(v_0, x_2) \leq \cdots \leq d(v_0, x_p)$.

That is, a higher indexed facility will be located further away from the supply plant on the optimal path. Moreover, since all facilities are located on the same path and there exists a unique path between any two points on a tree, we have

$$\sum_{i=j}^{k} d(x_i, x_{i+1}) = d(x_j, x_k) \qquad 1 \leq j < k \leq p. \tag{2.23}$$

Let v be a demand point node in T such that $v \notin P^*(v_0, x^*)$. Let x_j be the facility to which v is allocated and $w \geq 0$ be the demand at that node. For each demand point $v \in V$ we can split the cost of shipment from the supply plant up to the given demand point into two. Define $C(v)$ to be the cost of satisfying the demand at v. Then

$$C(v) = \alpha w d(v_0, x_1) + \alpha w \sum_{k=1}^{j-1} d(x_k, x_{k+1}) + w d(x_j, v). \tag{2.24}$$

Rearranging (2.24) we have

$$C(v) = \alpha w [d(v_0, x_1) + \sum_{k=1}^{j-1} d(x_k, x_{k+1})] + w d(x_j, v). \tag{2.25}$$

From (2.23) and $x_j \in P^*(v_0, x^*)$, we can rewrite (2.25) as follows:

$$C(v) = \alpha w d(v_0, x_j) + w d(x_j, v). \tag{2.26}$$

Let $v_P \in P^*(v_0, x^*)$ be a node such that $P(v_0, v_P) = P^*(v_0, x^*) \cap P(v_0, v)$. That is, v_P is the connection point of v to the path $P^*(v_0, x^*)$. We can now decompose the transportation cost in (2.26) into two parts:

$$C(v) \;=\; C_P(v) + C_{\bar{P}}(v),$$

where

$$\begin{aligned} C_P(v) &= \alpha w d(v_0, x_j) + w d(x_j, v_P), \\ C_{\bar{P}}(v) &= w d(v_P, v). \end{aligned}$$

$C_P(v)$ is the cost of transportation on the bulk shipment line which includes both the cost of a related bulk shipment and a portion of secondary transportation cost from facility j to demand point at node v. $C_{\bar{P}}(v)$ is the transportation cost on the non-bulk (or secondary) shipment route from the connection point on the optimal path to node v.

Observe that $C_{\bar{P}}(v)$ does not depend on the location of x_j on the path $P^*(v_0, x^*)$. Hence, if we know the path on which facilities must be located, we can simply add the weight w of all such nodes to their corresponding connection nodes on the optimal path and use the DP approach discussed in the previous section to solve the problem.

Since all the shipments originate from v_0, we can confine our search for the optimal path to only those that have v_0 as the starting point. Furthermore, since all demand points are located on the nodes of the network, application of the DP procedure to any path $P(v_0, x)$, $x \in T$ is exactly the same for the path $P(v_0, v_x)$ where v_x is the last node on the path from v_0 to x. Hence, it is sufficient to consider only those paths which start at v_0 and end at a node of the tree. There are exactly n such paths. In addition, one can show that the search can be further restricted to paths $P(v_0, v_t)$ where v_t is a leaf of the tree T. There can be at most $n-1$ leaves of a tree. This case occurs when the tree is a *star*, which is a tree in which all nodes except the root node are leaves (i.e. have degree one).

Chapter 3

A Single Assignment Branch-and-Bound Algorithm

In this chapter we present an implicit enumeration algorithm to solve SCLAP. Implicit enumeration methods are often used to solve NP-Hard combinatorial problems optimally. A Branch-and-Bound (B&B) type algorithm is one of the implicit enumeration techniques that is widely used. In this chapter, we will describe a single assignment B&B algorithm to solve SCLAP. In the single assignment approach, at each step just one facility is opened at a location point and the lower and upper bounds to the objective function including this assignment and previously located facilities are computed.

3.1 Problem Definition and Terminology

Let G be a transport network with node set $V = \{v_0, v_1, ..., v_n\}$ and edge set E. Each edge is considered to be an embedded edge (Dearing, Francis, and Lowe 1976) with a positive length and G is the union of all embedded edges. A point x in G is either a node or an interior point of some edge. For any two points x, y in G,

let the *distance*, denoted $d(x, y)$, be the length of a shortest path between x and y. Distance so defined has the usual properties of nonnegativity, symmetry, and triangle inequality of a metric (Rudin 1976). The supply plant is located at the node v_0 of the network. Note that if the supply plant is located on any other node of the network, the nodes of the network can be renumbered so that the supply plant is located at v_0. We call this node the supply node or the supply point. The demands occur at the remaining nodes of the network. We call those the demand nodes as well as the demand points.

Let $I = \{1, ..., n\}$ be the index set of the demand nodes of the network and $J = \{1, ..., p\}$ be the index set of facilities to be located. Let $\varphi : I \rightarrow J$ be a mapping and for a given $i \in I$, let $\varphi(i)$ denote the index of the facility to which demand point i is allocated. Let I_j be the index set of nodes which are allocated to facility j, i.e. $I_j = \{i \in I : \varphi(i) = j\}$ and $\cup_{j=1}^{p} I_j = I$. There is a positive demand, called *weight* and denoted w_i, associated with each demand node v_i, $i \in I$ of the network G. For ease of notation we will use $x_0 = v_0$ in the remaining part of the chapter. Now, the problem of interest is written as follows.

SCLAP: *Find the set of locations* $X = \{x_1, ..., x_p\} \subset V$ *on the network* G *and a partition of* I *into* p *subsets such that*

$$z = \alpha \left[\sum_{j=1}^{p} \sum_{i \in I_j} w_i \sum_{k=1}^{j} d(x_{k-1}, x_k) \right] + \sum_{j=1}^{p} \sum_{i \in I_j} w_i d(x_j, v_i) \tag{3.1}$$

is minimized.

$\sum_{k=1}^{j} d(x_{k-1}, x_k)$ gives the distance traveled on the supply route starting from the supply plant up to the j-th facility. The demand at a given demand point is first shipped to its designated facility from where it then receives its shipment

directly. This cost is multiplied by a factor α, $0 < \alpha < 1$, for interfacility or bulk shipment. Note that *no* direct shipments are allowed from the supply plant to the demand points. The supply route configuration and the distance traveled on the supply route change with respect to the location of facilities on the network. This is the routing decision portion of SCLAP.

Note that in this formulation it is assumed that the shipment vehicle does not return to the supply plant (e.g. pipeline system) or the cost of vehicle's empty return is negligible. However, it is simple to extend the problem to the case where the cost of empty return for the vehicle is significant: we simply add the cost $c\, d(x_p, v_0)$ to the objective function and the supply route becomes a closed loop instead of a path. The parameter c is defined to be the unit cost per distance for vehicle's return to supply plant empty after having served all facilities and $d(x_p, v_0)$ is the distance from the last facility visited to the supply plant located at v_0.

Let us rewrite the first portion, which corresponds to the transportation cost on the supply route, of (3.1) in an open form. Before we do that, for ease of notation let $W_j = \sum_{i \in I_j} w_i$ denote the total amount of demand allocated to a given facility. Now we have

$$\begin{aligned}
\alpha \sum_{j=1}^{p} W_j \sum_{k=1}^{j} d(x_{k-1}, x_k) \;=\; & \alpha W_1 d(x_0, x_1) + \\
& \alpha W_2 \left[d(x_0, x_1) + d(x_1, x_2) \right] + \\
& \alpha W_3 \left[d(x_0, x_1) + d(x_1, x_2) + d(x_2, x_3) \right] + \\
& \vdots \\
& \alpha W_p \left[d(x_0, x_1) + d(x_1, x_2) + \cdots + d(x_{p-1}, x_p) \right]. \quad (3.2)
\end{aligned}$$

Reorganizing RHS of (3.2) gives:

$$\alpha \sum_{j=1}^{p} W_j \sum_{k=1}^{j} d(x_{k-1}, x_k) = \left(\sum_{k=1}^{p} \alpha W_k \right) d(x_0, x_1) + \left(\sum_{k=2}^{p} \alpha W_k \right) d(x_1, x_2) + \left(\sum_{k=3}^{p} \alpha W_k \right) d(x_2, x_3) + \cdots + \alpha W_p d(x_{p-1}, x_p) \quad (3.3)$$

in closed form, we have

$$\alpha \sum_{j=1}^{p} W_j \sum_{k=1}^{j} d(x_{k-1}, x_k) = \sum_{j=1}^{p} \sum_{k=j}^{p} \alpha W_k d(x_{j-1}, x_j). \quad (3.4)$$

Inserting (3.4) into (3.1) we get an equivalent formulation of the objective function of SCLAP as follows:

$$z = \alpha \sum_{j=1}^{p} \sum_{k=j}^{p} (\sum_{i \in I_k} w_i) d(x_{j-1}, x_j) + \sum_{j=1}^{p} \sum_{i \in I_j} w_i d(x_j, v_i). \quad (3.5)$$

In this formulation we see another way to look at the problem: i.e. all the demand in the system is shipped to the first facility on the supply route, the portion of total demand that is allocated to this facility is unloaded here and the rest of the demand is shipped to the second facility, the demand allocated to that facility is unloaded there and the rest of the material is sent to the third facility on the supply route, and so on.

3.2 Bounds for SCLAP

In SCLAP, supply plant x_0 originates all the material to be distributed in the system. Therefore, for a given demand point in the system it is profitable to receive its shipment via some facility (transshipment point) only when the cost of shipping the material through the supply route is less than the cost of receiving the shipment

directly from the supply plant itself (assuming direct shipments from the supply plant were allowed). This is achieved by discounting the cost of transportation on the supply route by the factor α as discussed in Chapters 1 and 2. In other words, if we were to ship all the material originating in the supply plant directly to demand points along the shortest path routes with the discounted rate, no other facilities would be used as transshipment points and all of them would be located on top of the supply plant. Although in the original problem direct shipments from the supply plant are not allowed, clearly, this arrangement provides a lower bound on the optimal objective value z^* of (3.1). The following theorem defines this lower bound on SCLAP.

Theorem 3.1

$$\underline{z} = \alpha \sum_{i=1}^{n} w_i d(x_0, v_i)$$

gives a lower bound for SCLAP.

In the remaining part of this section we will give lower and upper bounds for SCLAP under the condition that we know the locations of first K $(K \leq p)$ facilities on the supply route. Those bounds will be used in the Branch-and-Bound algorithm described in the next section. But before we do that we will define the optimal allocation solution when the locations of facilities are known.

Since we want to minimize the transportation cost and there are no capacity constraints on facilities, demand points are allocated to the facility which provides the least unit cost of transportation for their shipment. If there are ties, they can be broken arbitrarily so that each demand point is allocated to one facility only. We restate this observation more formally below.

Theorem 3.2 *Let $x_1^*, ..., x_p^*$ be the fixed locations of facilities. The facilities are visited in the given order on the supply route. Then, a given demand point i is allocated to a facility j^* which gives*

$$\min_{1 \le j \le p} \alpha \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + d(x_j^*, v_i);$$

ties are broken arbitrarily.

Proof: Since there are no capacity constraints on facilities, it is clear that in the optimal solution each demand point can be allocated to the one facility which minimizes the transportation cost. Let $j(i)$, $i = 1, ..., n$, denote the facility to which demand point i is allocated. Then the objective becomes

$$\sum_{i=1}^{n} w_i \left(\alpha \sum_{k=1}^{j(i)} d(x_{k-1}^*, x_k^*) + d(x_{j(i)}^*, v_i) \right). \tag{3.6}$$

To minimize this function, we have to find the facility indices $j(i)$ for each demand point i. Since $w_i \ge 0$, $\forall i$, it is clear that minimizing the above function is equivalent to finding

$$\sum_{i=1}^{n} w_i \left(\min_{1 \le j \le p} \alpha \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + d(x_j^*, v_i) \right). \tag{3.7}$$

The portion of (3.7) within the big parentheses gives us what is stated in Theorem 3.2. That is, to minimize total transportation cost each demand point i is allocated to a facility which minimizes the cost of transporting one unit of demand from supply plant to the given demand point. □

This theorem provides us with an efficient procedure to find the optimal allocations when the locations of facilities and the route (not necessarily optimal) are known. Since the sums $\alpha \sum_{k=1}^{j} d(x_{k-1}, x_k)$ $1 \le j \le p$ are independent of the demand

points we can compute them beforehand in $O(p)$ time and then for each demand point we can find the index of the facility which gives

$$\min_{1\leq j\leq p} \alpha \sum_{k=1}^{j} d(x_{k-1}, x_k) + d(x_j, v_i)$$

in $O(p)$ time. When we do this for all demand points, then the computational complexity is $O(np)$. The overall computational time is $O(p) + O(np)$ which is also $O(np)$.

Suppose now that we know the locations of $x_1^*, ..., x_K^*$ of the first K ($K \leq p$). which are visited according to the sequence. $1, 2, ..., K$. facilities. Assume that the shipments outside the supply route could also be done with the discounted rate. Then all demand points would be allocated to the supply plant because now it provides direct shipments with the discounted rate. However, in order to get a tighter lower bound, we will use the rule described in Theorem 3.2 for allocating demand points to facilities over K facilities. However, we calculate the transportation costs with the "everywhere discounted" rate and we show that such configuration provides a lower bound on the optimal objective function value of SCLAP.

Let the first K ($K \leq p$) facilities on the supply route be located at $x_1^*, ..., x_K^*$. Define $z(K)$ to be the optimal objective function value to SCLAP with the locations of the first K facilities and their positions on the supply route fixed. Let $X^* = \{x_1^*, .., x_K^*, ., x_p^*\}$ be the given locations of all facilities. We know that each demand point is allocated to a facility according to the allocation rule of Theorem 3.2. In this way we find the new allocation configurations for p facilities. When the new allocation sets are constructed, a demand point will either be allocated to the same facility in the optimal solution or it will be allocated to another facility j^*,

$j^* \in \{K+1, ..., p\}$. Clearly, if a demand point changes its facility in the optimal solution, it can only be one of the new facilities because according to the allocation rule of Theorem 3.2 if there has been another facility in the first K facilities which gave a lesser cost it would have already been chosen.

For the first case, when a demand point i is allocated to the same facility in the optimal location-allocation and a partial location-allocation solution, it is simple to show that the cost of transportation, for that demand point, is less than or equal to the cost of transportation in the optimal solution. By partial location-allocation solution we mean that only $K < p$ facilities are located and demand points are allocated to those K facilities. We rephrase this in the following observation.

Observation 3.1 *Let $X^* = \{x_1^*, ..., x_K^*, ., x_p^*\}$ be the given locations of all facilities, where facilities are indexed according to the visiting sequence on the supply route. Then*

$$\alpha w_i \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + \alpha w_i d(x_j^*, v_i) \leq \alpha w_i \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + w_i d(x_j^*, v_i). \qquad (3.8)$$

Proof: Since $0 < \alpha < 1$, (3.8) is true. □

In the second case (when the facility to which demand point i is allocated changes in the optimal solution), we have to show that the cost for demand point i calculated in the partial location-allocation solution is less than or equal to the cost calculated in the p-facility solution. We state this in the next observation

Observation 3.2 *Let $X^* = \{x_1^*, .., x_K^*, ., x_p^*\}$ be the given locations of all facilities, where facilities are indexed according to the visiting sequence on the supply route,*

and let j' denote the index of the facility allocation for demand point i in the partial location-allocation solution, and j^* be the index of the facility in the p-facility solution. Then

$$\alpha w_i \sum_{k=1}^{j'} d(x^*_{k-1}, x^*_k) + \alpha w_i d(x^*_{j'}, v_i) \leq \alpha w_i \sum_{k=1}^{j^*} d(x^*_{k-1}, x^*_k) + w_i d(x^*_{j^*}, v_i). \tag{3.9}$$

Proof: Since $w_i \geq 0$ we can drop w_i from both sides of the inequality. Since we know that $j^* > K \geq j'$, we can rewrite the RHS of the inequality (3.9) as follows:

$$\alpha \sum_{k=1}^{j^*} d(x^*_{k-1}, x^*_k) + d(x^*_{j^*}, v_i) = \alpha \sum_{k=1}^{j'} d(x^*_{k-1}, x^*_k) + \alpha \sum_{k=j'+1}^{j^*} d(x^*_{k-1}, x^*_k) + d(x^*_{j^*}, v_i). \tag{3.10}$$

From the triangle inequality property of distance we know that

$$d(x^*_{j'}, v_i) \leq \sum_{k=j'+1}^{j^*} d(x^*_{k-1}, x^*_k) + d(x^*_{j^*}, v_i). \tag{3.11}$$

i.e. any "detour" in the new configuration will cost more. Since $\alpha > 0$, this gives

$$\begin{aligned} \alpha \sum_{k=1}^{j'} d(x^*_{k-1}, x^*_k) + \alpha d(x^*_{j'}, v_i) &\leq \alpha \sum_{k=1}^{j'} d(x^*_{k-1}, x^*_k) + \alpha \left[\sum_{k=j'+1}^{j^*} d(x^*_{k-1}, x^*_k) + d(x^*_{j^*}, v_i) \right] \\ &= \alpha \sum_{k=1}^{j^*} d(x^*_{k-1}, x^*_k) + \alpha d(x^*_{j^*}, v_i) \\ &\leq \alpha \sum_{k=1}^{j^*} d(x^*_{k-1}, x^*_k) + d(x^*_{j^*}, v_i). \end{aligned} \tag{3.12}$$

The LHS of (3.12) is the transportation cost for a given demand point calculated with the "everywhere" discounted rate by the partial location allocation solution and

the RHS is the transportation cost calculated by the p-facility solution. Since $w_i \geq 0$ (3.12) is equivalent to (3.9). Thus the proof is complete. □

Observations 3.1 and 3.2 lead to the following theorem which guarantees a lower bound for the optimal objective function value when the optimal locations of first K facilities are known.

Theorem 3.3 *Let $x_1^*, ..., x_K^*$ be the locations of first K ($K \leq p$) facilities on the supply route. Let I_j', $1 \leq j \leq K$ give the sets of demand point indices which are allocated to facilities according to the rule given in Theorem 3.2 over K open facilities. Then*

$$\underline{z}(K) = \alpha \sum_{j=1}^{K} \sum_{i \in I_j'} w_i \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + \alpha \sum_{j=1}^{K} \sum_{i \in I_j'} w_i d(x_j^*, v_i)$$

gives a lower bound on the optimal objective function value $z(K)$ of SCLAP with first K facilities located at $x_1^*, ..., x_K^*$.

Proof: For each demand point v_i either Case 1 or Case 2 occurs as described earlier. It is shown in Observations 3.1 and 3.2 that, in either case, the cost of transportation for the given demand point in the partial allocation solution with the "everywhere discounted" rate will be less than or equal to the cost of transportation in the optimal solution with the locations of first K facilities as given. Summation of the costs with the "everywhere discounted" rate in the the partial allocation, over all demand points gives $\underline{z}(K)$. Thus $\underline{z}(K)$ is a lower bound for the optimal objective function value of SCLAP with first K facilities located at $x_1^*, ..., x_K^*$. □

Theorem 3.3 gives a lower bound to the objective value of SCLAP with respect to a preset partial location selection. Clearly, if the location of the first K

facilities are optimal with respect to SCLAP, this theorem will give a lower bound on the optimal objective function value of SCLAP.

Similarly, we will define upper bounds on the optimal objective function value by allocating the demand points according to the partial location selection given. This time the costs are calculated with no discount outside the supply route.

Theorem 3.4 *Let $x_1^*, ..., x_K^*$ be the locations of first K ($K \leq p$) facilities on the supply route. Let I_j', $1 \leq j \leq K$ give the sets of demand point indices which are allocated to facilities according to the rule given in Theorem 3.2 over K open facilities. Then*

$$\overline{z}(K) = \alpha \sum_{j=1}^{K} \sum_{i \in I_j'} w_i \sum_{k=1}^{j} d(x_{k-1}^*, x_k^*) + \sum_{j=1}^{K} \sum_{i \in I_j'} w_i d(x_j^*, v_i)$$

gives an upper bound on the optimal objective function value of SCLAP.

Proof: To prove Theorem 3.4, it is sufficient to show that there exists at least one demand point whose transportation cost $z(K)$, in the p-facility solution containing first K facilities, is less than or equal to the cost calculated in Theorem 3.4. Let v_s be a demand point which is allocated to the K-th facility in the partial location-allocation solution. Let the new facility x_{K+1}^* be located on v_s, i.e. $x_{K+1}^* = v_s$. Then we have

$$\alpha w_s \sum_{k=1}^{K} d(x_{k-1}^*, x_k^*) + w_s d(x_K^*, v_s) \geq \alpha w_s \sum_{k=1}^{K} d(x_{k-1}^*, x_k^*) + \alpha w_s d(x_K^*, v_s). \qquad (3.13)$$

Since $x_{K+1}^* = v_s$, RHS of (3.13) is equal to $\alpha w_s \sum_{k=1}^{K+1} d(x_{k-1}^*, x_k^*)$ which is also equal to $\alpha w_s \sum_{k=1}^{K+1} d(x_{k-1}^*, x_k^*) + \alpha w_i d(x_{K+1}^*, v_s)$ because $d(x_{K+1}^*, v_s) = 0$. Thus we have

$$\alpha w_s \sum_{k=1}^{K} d(x_{k-1}^*, x_k^*) + w_s d(x_K^*, v_s) \geq \alpha w_s \sum_{k=1}^{K+1} d(x_{k-1}^*, x_k^*) + w_s d(x_{K+1}^*, v_s) \qquad (3.14)$$

which states that for the demand node v_s the cost of transportation calculated according to Theorem 3.4 is greater than or equal to the cost calculated in the p-facility allocation. Therefore $\overline{z}(K)$ gives an upper bound on the p-facility objective function value $z(K)$ of SCLAP. Note that $\overline{z}(K)$ is an upper bound for the p-facility location solution with the first K facilities fixed anywhere and SCLAP is a minimization problem. Therefore, $\overline{z}(K)$ will be greater than or equal to the objective value of the p-facility solution when the locations of the first K facilities are optimal with respect to SCLAP, in which case the p-facility solution *is* the optimal solution to SCLAP. Thus, $\overline{z}(K)$ is an upper bound on the optimal objective function value of SCLAP. □

3.3 A Single Assignment Branch & Bound Algorithm for Solving SCLAP

Let P represent a partial location selection indicating that $1, .., |P|$-th facilities are located. P is an ordered set of $|P|$ facilities where $|P| \leq p$. That is, the node index in the j-th position of the set P indicates that the j-th facility is located on that node. With each partial selection P we associate a lower bound $\underline{z}(P)$ and an upper bound, $\overline{z}(P)$, which can be calculated by the methods described in the previous section.

To describe the B&B scheme for the single assignment algorithm, let $\bar{z}$ denote the least upper bound on the optimal objective function value of SCLAP calculated so far. We start with the partial selection corresponding to $P = \emptyset$. If, for a given partial selection, $\underline{z}(P) \leq \bar{z}$ holds, then we choose an index $i \in I - P$ and replace P by $P \cup \{i\}$, i.e.

$$P \leftarrow P \cup \{i\}.$$

Then we derive new lower and upper bounds $\underline{z}(P)$, $\overline{z}(P)$ for the updated P by applying the procedures described in the previous section. If $\overline{z}(P) \leq \bar{z}$ then $\bar{z}$ is replaced by $\bar{z}(P)$, i.e.

$$\bar{z} \leftarrow \bar{z}(P).$$

If $|P| = p$ and $\bar{z}(P) \leq \bar{z}$, then this newly found solution is stored as a possible candidate for the optimal solution and $\bar{z}$ is improved.

If $\underline{z}(P) \geq \bar{z}$, one returns to the last found partial selection P with $\underline{z}(P) < \bar{z}$. The assignment of facility j to be located on node i (denoted $j \rightsquigarrow i$), chosen at the transition of P to its successor, is now blocked.

A favorable selection of an index i for a new branching can be made by selecting the node which gives the smallest upper bound.

We next illustrate the above B&B Algorithm by a numerical example.

3.3.1 Example

The example network is given in Figure 3.1 and the corresponding cost data is given in Figure 3.2. The discounting factor α is set to 0.50 and the number of facilities to locate, p, is set to 2. The facilities are to be located on the nodes of the network.

For $P = \emptyset$ we have $\underline{z} = \alpha \sum_{i=1}^{5} w_i d(v_0, v_i) = 31.5$ by Theorem 3.1. Initially, define $\bar{z} \equiv \infty$ (in practice, of course, $\bar{z}$ is set to a very large number). We now calculate the upper bound $\bar{z}(i)$ for each of the nodes given the partial set of locations according to Theorem 3.4. Then we get

$$\bar{z}(1) = 62$$

$$\bar{z}(2) = 75$$

$$\bar{z}(3) = 86$$

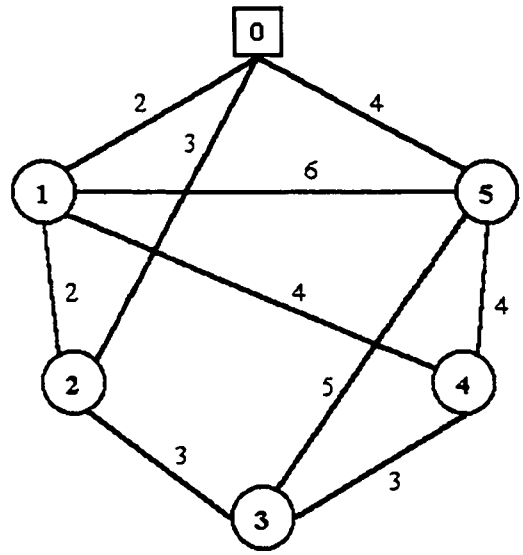

Figure 3.1: Diagram of the Example Problem Network

Edge Lengths

	0	1	2	3	4	5
0	0	2	3	-	-	4
1		0	2	-	4	6
2			0	3	-	-
3				0	3	5
4					0	4
5						0

Distances

	0	1	2	3	4	5
0	0	2	3	6	6	4
1	2	0	2	5	4	6
2	3	2	0	3	6	7
3	6	5	3	0	3	5
4	6	4	6	3	0	4
5	4	6	7	5	4	0

Weights

1	2	3	4	5
2	3	2	5	2

Figure 3.2: Data for the Example Problem

$$\bar{z}(4) = 82$$

$$\bar{z}(5) = 91$$

Since $\bar{z}(1) = 62$ is the smallest one, we choose the single assignment $x_1 = v_1$ for the next branching. Also, since $\bar{z}(1) = 62 < \bar{z}_{current} = \infty$ we update the current upper bound.

Next we move to branch $(P = (1))$ and calculate the lower bound by Theorem 3.3, which gives:

$\underline{z}(1) = 38$.

Since $\underline{z}(1) \leq \bar{z}$ we continue branching. Next we calculate the upper bounds for $\bar{z}(1, i)$ as follows:

$$\bar{z}(1, 2) = 57$$

$$\bar{z}(1, 3) = 57$$

$$\bar{z}(1, 4) = 52$$

$$\bar{z}(1, 5) = 56$$

Since $\bar{z}(1, 4) = 52$ is the smallest we choose the single assignment $2 \rightsquigarrow 4$ for the next branching. Now we have $|P| = 2$, then the upper bound is now the actual objective function value for the given permutation of facility locations. Since $\bar{z}(1, 4) = 52 < \bar{z}_{current} = 62$ we update the current upper bound as well as keep the permutation as a candidate solution. Since we have calculated all possible values for facility 2 given facility 1 is located at node 1 and selected the minimum, we don't have to branch further down on the node $(P = (1), 2 \not\rightsquigarrow 4)$. Thus this branch is blocked.

Next we branch at $(P = \emptyset, 1 \not\rightsquigarrow 1)$. We select the next smallest value of $\bar{z}(i)$ which we have already calculated in the first branching. For the node $(P = (2))$ we calculate the lower bound which is $\underline{z}(2) = 48$ This lower bound is not greater than

the current upper bound, $\bar{z} = 50$, therefore we have to continue on this branch. Next we calculate the upper bounds $\bar{z}(2, i)$ as follows:

$$\bar{z}(2, 1) = 68$$

$$\bar{z}(2, 3) = 63.5$$

$$\bar{z}(2, 4) = 60$$

$$\bar{z}(2, 5) = 64$$

Note that, since now $|P| = 2 = p$, the upper bounds calculated here are also the actual objective function values for the given candidate solutions. However, we have already found a candidate solution, namely $P = (1, 4)$ whose objective function value is less than the values calculated here. Therefore, the current branch is now blocked.

Next we branch at $(P = \emptyset, 1 \not\rightarrow 2)$. The rest of the procedure follows in a similar fashion. Figure 3.3 gives the entire B&B Tree.

The optimal solution locates first facility on node 1 and second facility on node 4, i.e. $x_1 = v_1$ and $x_2 = v_4$. And the facilities are visited as x_1 being the first and x_2 being the second. Demand nodes v_1, v_2, v_3, and v_5 are allocated to the first facility located at $x_1 = v_1$ and demand point v_4 is allocated the second facility located at $x_2 = v_4$. An alternate optimal solution locates the facilities at the same locations with demand point v_3 being allocated to facility 2 instead of facility 1; all other allocations are the same.

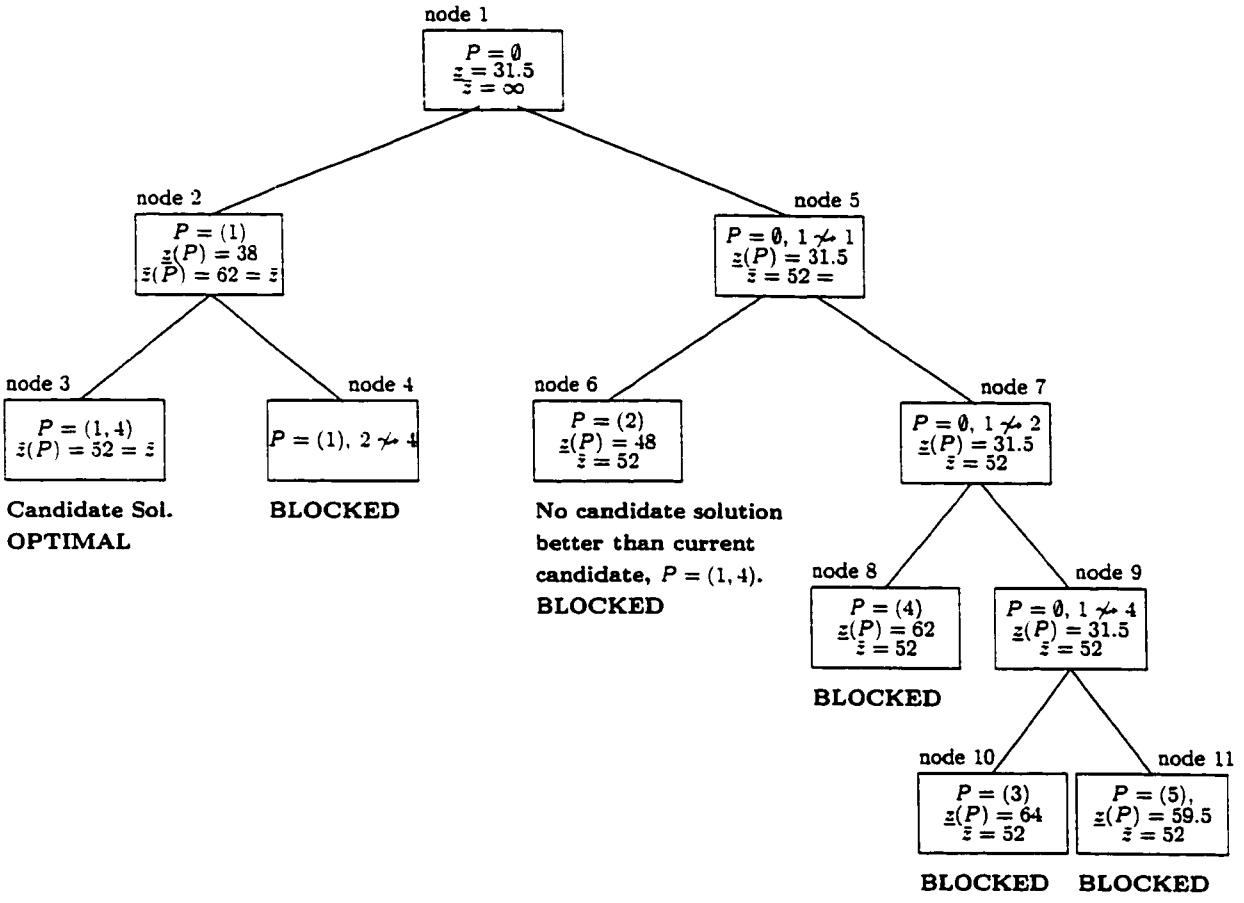

Figure 3.3: B&B Tree of the Example Problem

3.4 Computational Results

In this section we present the results for the computational experiments done on randomly generated networks. We have generated 10 different problems for each of the selected number of demand points (n=10,20,...,50). The weights for the demand points and the edge lengths are generated randomly from a uniform distribution between 1 and 100. We have solved these problems with 3 and 5 facilities to locate and also observed problem behavior for different α values. All the problems were solved on a Sun Sparc V. with uP 70MHz Processor.

Tables 3.1-3.6. gives the processing times (in CPU seconds) of all the problems solved for different n, p, α combinations. The last column of Tables 3.2, 3.4, 3.6, gives the average CPU time over 10 problems. Note that 0.0 means the algorithm was able to solve all 10 problems in less than one second of CPU time.

Since the number of branches of the B&B tree increase exponentially with n and p, the processing times increase rapidly as n or p increases. One interesting point to note is the decrease in processing time as α moves from 0 to 1. This is not unexpected since, as α gets closer to 1, the gap between the lower and upper bounds becomes smaller, allowing more branches to be pruned early on. The computational difficulty of the problem is clearly indicated when there are 30 demand points and 5 facilities to locate. The average processing time took a little less than an hour of CPU time for $\alpha = 0.2$. In all cases, it is seen that the value of α plays a significant role in the computational difficulty of the problems. Figure 3.4 illustrates this effect clearly, for the example problem with n=20 demand points and p=5 facilities to locate. In most practical cases however, it is more likely that α is closer to 0 than to

n	p	α	CPU (in seconds)				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	pr. 5
10	3	0.1	0	0	0	0	1
10	3	0.2	0	0	0	0	0
10	3	0.3	0	0	0	0	0
10	3	0.4	0	0	0	0	0
10	3	0.5	0	0	0	0	0
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	0	0	0
10	3	0.8	0	0	0	0	0
10	3	0.9	0	0	0	0	0
10	5	0.1	2	2	2	1	1
10	5	0.2	2	2	1	2	1
10	5	0.3	0	2	1	0	1
10	5	0.4	1	1	1	1	1
10	5	0.5	1	1	1	0	1
10	5	0.6	0	1	0	1	1
10	5	0.7	1	1	1	0	0
10	5	0.8	0	0	0	0	1
10	5	0.9	0	1	1	1	0

Table 3.1: Processing Times of B&B Algorithm for 10-node Examples

n	p	α	CPU (in seconds)					Avg.
			Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10	CPU
10	3	0.1	0	0	0	0	0	0.1
10	3	0.2	0	0	0	0	0	0.0
10	3	0.3	0	0	0	0	0	0.0
10	3	0.4	0	0	0	0	0	0.0
10	3	0.5	0	0	0	0	0	0.0
10	3	0.6	0	0	0	0	0	0.0
10	3	0.7	0	0	0	0	0	0.0
10	3	0.8	0	0	0	0	0	0.0
10	3	0.9	0	0	0	0	0	0.0
10	5	0.1	2	2	3	3	2	2.0
10	5	0.2	2	1	1	2	1	1.5
10	5	0.3	1	2	1	2	1	1.1
10	5	0.4	1	1	1	1	1	1.0
10	5	0.5	1	1	0	1	1	0.8
10	5	0.6	0	0	1	1	1	0.6
10	5	0.7	0	1	0	0	0	0.4
10	5	0.8	1	0	0	1	1	0.4
10	5	0.9	0	0	0	0	0	0.3

Table 3.2: Processing Times of B&B Algorithm for 10-node Examples (continued)

n	p	α	CPU (in seconds)				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	pr. 5
20	3	0.1	0	0	0	1	1
20	3	0.2	1	1	1	0	0
20	3	0.3	0	1	0	1	1
20	3	0.4	0	0	1	0	0
20	3	0.5	0	0	0	1	0
20	3	0.6	1	0	0	0	0
20	3	0.7	0	1	0	0	0
20	3	0.8	0	0	0	0	1
20	3	0.9	0	0	0	0	0
20	5	0.1	182	226	232	215	208
20	5	0.2	144	184	160	191	173
20	5	0.3	95	123	94	120	144
20	5	0.4	79	94	70	107	118
20	5	0.5	67	87	53	89	72
20	5	0.6	34	63	32	64	33
20	5	0.7	33	34	22	44	11
20	5	0.8	23	33	12	22	13
20	5	0.9	23	11	12	12	11

Table 3.3: Processing Times of B&B Algorithm for 20-node Examples

n	p	α	CPU (in seconds)					Avg.
			Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10	CPU
20	3	0.1	0	1	1	0	0	0.4
20	3	0.2	1	1	0	1	1	0.7
20	3	0.3	0	0	1	0	0	0.4
20	3	0.4	1	0	0	1	1	0.4
20	3	0.5	0	1	0	0	0	0.2
20	3	0.6	0	0	0	0	0	0.1
20	3	0.7	0	0	0	1	0	0.2
20	3	0.8	0	0	1	0	0	0.2
20	3	0.9	0	0	0	0	0	0.0
20	5	0.1	220	230	211	210	202	213.6
20	5	0.2	182	175	143	165	132	164.9
20	5	0.3	156	117	112	143	112	121.6
20	5	0.4	117	67	88	137	108	98.5
20	5	0.5	64	61	78	103	102	77.6
20	5	0.6	46	57	68	67	68	53.2
20	5	0.7	45	32	45	61	55	38.2
20	5	0.8	34	23	46	41	33	28.0
20	5	0.9	22	22	22	11	11	15.7

Table 3.4: Processing Times of B&B Algorithm for 20-node Examples (continued)

n	p	α	CPU (in seconds)				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	pr. 5
30	3	0.2	3	3	3	3	2
30	3	0.4	1	1	1	2	1
30	3	0.6	1	1	1	0	1
30	3	0.8	0	0	0	1	0
30	5	0.2	2431	2074	2329	2000	2476
30	5	0.4	1016	823	1594	1142	1411
30	5	0.6	573	229	359	720	499
30	5	0.8	285	191	193	194	279
40	3	0.2	8	10	10	9	9
40	3	0.4	5	3	3	7	4
40	3	0.6	1	2	2	2	2
40	3	0.8	1	0	0	1	0
50	3	0.2	23	22	23	23	23
50	3	0.4	15	11	10	11	8
50	3	0.6	5	4	5	3	3
50	3	0.8	3	1	1	1	1

Table 3.5: Processing Times of B&B Algorithm for 30,40,50-node Examples

n	p	α	CPU (in seconds) Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10	Avg. CPU
30	3	0.2	3	3	3	3	3	2.9
30	3	0.4	2	1	2	1	1	1.3
30	3	0.6	1	0	1	1	1	0.8
30	3	0.8	0	1	0	0	0	0.2
30	5	0.2	2364	1900	2316	2116	2225	2223.1
30	5	0.4	1162	854	1570	1123	806	1150.1
30	5	0.6	675	481	869	385	289	507.9
30	5	0.8	191	192	288	291	187	229.1
40	3	0.2	10	9	9	9	9	9.2
40	3	0.4	5	4	4	5	5	4.5
40	3	0.6	2	3	1	2	2	1.9
40	3	0.8	1	1	0	1	1	0.6
50	3	0.2	24	20	22	24	23	22.7
50	3	0.4	15	12	13	12	17	12.4
50	3	0.6	2	3	4	2	5	3.6
50	3	0.8	1	2	2	1	1	1.4

Table 3.6: Processing Times of B&B Algorithm for 30,40,50-node Examples (continued)

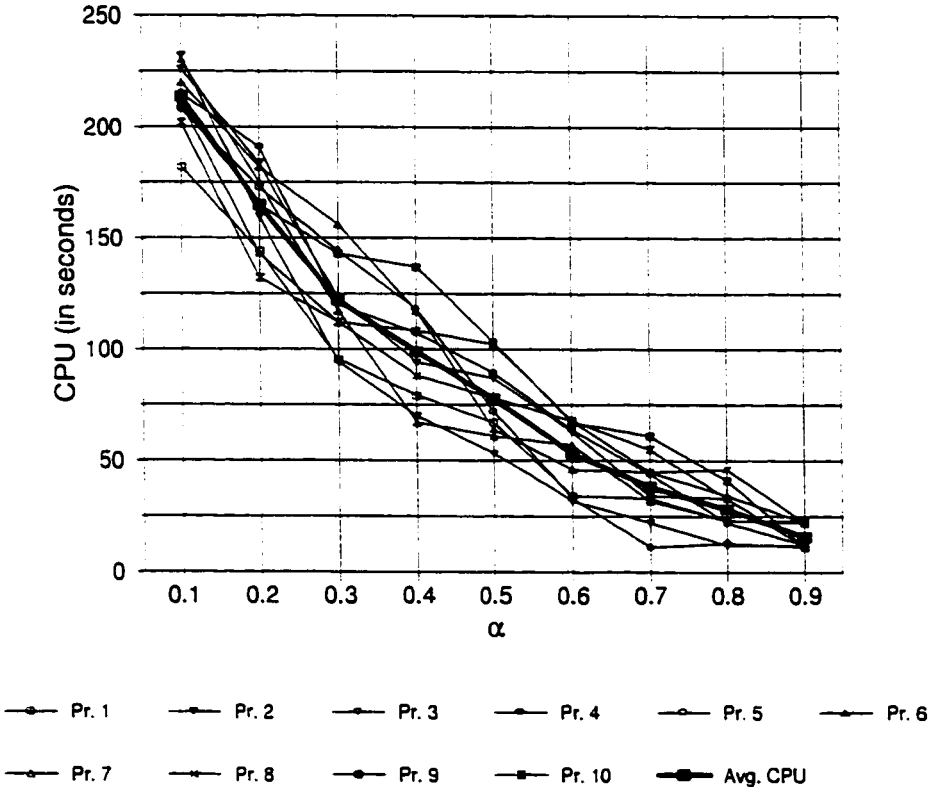

Figure 3.4: Effect of α on Processing Time

1, because it is generally expected that the cost of transportation on a "supply route" be significantly less than the cost of transporting the goods directly to each demand point.

Next, Tables 3.7-3.12 gives the number of branching nodes in the B&B tree for all problems solved. The processing time of the B&B Algorithm is directly proportional to the number of branches in the B&B tree. That is, the processing time is the number of branches multiplied by the computation time at each branch. The computation time at each branch is is $O(n^2p)$; the lower and upper bounds are calculated in $O(np)$ time for each candidate site and the site which gives the smallest upper bound is selected as the next branch. The size of the branch and bound tree increases exponentially as the size of the problem increases, for given values of n and p the size of the B&B tree would be $2^{P_p^n}$, where P_p^n represents the Permutation$(n,p) = \frac{n!}{(n-p)!}$, if all the branches were traversed. Because of the direct relation between the processing time and the size of the B&B tree, as one would expect, Tables 3.7-3.12 provide results in direct relation with the processing times.

n	p	α	Number of Branch Nodes				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	Pr. 5
10	3	0.1	194	202	202	202	202
10	3	0.2	182	182	146	122	142
10	3	0.3	102	164	102	122	84
10	3	0.4	62	120	62	84	64
10	3	0.5	62	82	62	62	62
10	3	0.6	42	82	62	42	62
10	3	0.7	42	62	44	22	62
10	3	0.8	22	42	42	22	42
10	3	0.9	22	42	42	22	22
10	5	0.1	9634	9332	7716	7380	7322
10	5	0.2	6490	8474	5694	5008	4902
10	5	0.3	4056	7430	5638	3548	3976
10	5	0.4	3350	5440	3418	3000	3380
10	5	0.5	2346	4536	3504	1838	3448
10	5	0.6	2318	4508	3448	1174	3490
10	5	0.7	2262	2346	2292	1174	3504
10	5	0.8	1174	2346	2332	1174	2218
10	5	0.9	1174	2346	1174	1174	1146

Table 3.7: Number of Nodes in the B&B Tree for 10-node Examples

n	p	α	Number of Branch Nodes				
			Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10
10	3	0.1	202	202	202	202	202
10	3	0.2	142	142	182	202	138
10	3	0.3	122	122	158	202	102
10	3	0.4	82	82	84	148	102
10	3	0.5	62	62	42	142	62
10	3	0.6	42	42	22	84	62
10	3	0.7	22	22	22	42	62
10	3	0.8	22	22	22	42	42
10	3	0.9	22	22	22	22	42
10	5	0.1	8966	8032	10142	11490	6952
10	5	0.2	7094	7062	7134	9710	5430
10	5	0.3	5726	6536	4300	7252	5114
10	5	0.4	3518	4452	2818	6158	4008
10	5	0.5	3420	3434	2074	4072	3504
10	5	0.6	2262	2346	1174	3504	3518
10	5	0.7	1174	2346	1174	2346	3360
10	5	0.8	1174	2346	1174	2034	2346
10	5	0.9	1174	1174	1174	1174	2318

Table 3.8: Number of Nodes in the B&B Tree for 10-node Examples (continued)

n	p	α	Number of Branch Nodes				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	Pr. 5
20	3	0.1	802	802	802	802	802
20	3	0.2	642	800	802	740	682
20	3	0.3	446	648	584	722	602
20	3	0.4	322	446	322	442	442
20	3	0.5	282	322	202	402	282
20	3	0.6	242	284	162	322	122
20	3	0.7	122	164	82	162	82
20	3	0.8	82	122	82	82	42
20	3	0.9	82	42	42	42	42
20	5	0.1	197626	246426	246030	231488	228144
20	5	0.2	154992	202996	174462	203622	188452
20	5	0.3	102140	133830	103748	131448	155152
20	5	0.4	86466	100808	77530	117500	127048
20	5	0.5	73470	94768	57234	98564	80070
20	5	0.6	37058	69692	34298	69588	36378
20	5	0.7	36820	37058	24706	45084	12354
20	5	0.8	24706	36956	12354	24706	12354
20	5	0.9	24706	12354	12354	12354	12354

Table 3.9: Number of Nodes in the B&B Tree for 20-node Examples

n	p	α	Number of Branch Nodes				
			Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10
20	3	0.1	802	802	802	802	802
20	3	0.2	762	764	686	762	656
20	3	0.3	642	600	482	602	448
20	3	0.4	524	282	402	522	402
20	3	0.5	284	242	282	478	362
20	3	0.6	162	202	282	242	322
20	3	0.7	162	162	162	242	202
20	3	0.8	122	82	162	162	162
20	3	0.9	82	82	82	42	42
20	5	0.1	238338	244962	228038	229178	217252
20	5	0.2	200326	189010	153506	181362	145226
20	5	0.3	171404	128546	121736	155196	123464
20	5	0.4	126694	74080	95928	147784	118890
20	5	0.5	69452	66344	85718	111666	111000
20	5	0.6	49410	61660	75218	74114	73774
20	5	0.7	49206	34504	49410	67196	60570
20	5	0.8	37058	24706	48900	44808	36072
20	5	0.9	24638	24706	24706	12354	12354

Table 3.10: Number of Nodes in the B&B Tree for 20-node Examples (continued)

n	p	α	Number of Branch Nodes				
			Pr. 1	Pr. 2	Pr. 3	Pr. 4	Pr. 5
30	3	0.2	1800	1624	1742	1742	1682
30	3	0.4	888	784	848	906	1084
30	3	0.6	422	182	242	482	422
30	3	0.8	244	122	122	122	182
30	5	0.2	1128510	961426	1093968	945548	1096006
30	5	0.4	476596	383370	479078	544002	667196
30	5	0.6	273140	107188	169156	341714	238280
30	5	0.8	136544	91066	91066	91066	132436
40	3	0.2	2882	3202	3146	3042	3042
40	3	0.4	1530	1204	1124	2166	1528
40	3	0.6	482	486	402	728	400
40	3	0.8	242	162	82	242	162
50	3	0.2	5002	4900	4902	4902	4900
50	3	0.4	3202	2402	2212	2406	1816
50	3	0.6	1204	904	904	710	608
50	3	0.8	502	202	304	202	302

Table 3.11: Number of Nodes in the B&B Tree for 30,40,50-node Examples

n	p	α	Number of Branch Nodes				
			Pr. 6	Pr. 7	Pr. 8	Pr. 9	Pr. 10
30	3	0.2	1682	1622	1680	1680	1742
30	3	0.4	974	670	1142	782	610
30	3	0.6	422	362	662	242	182
30	3	0.8	122	122	182	182	122
30	5	0.2	1095478	892288	1095618	992922	1029896
30	5	0.4	540324	396744	745854	522754	376288
30	5	0.6	318726	227662	409304	182010	136598
30	5	0.8	91066	91066	136598	136598	89608
40	3	0.2	3122	3042	3122	3124	3122
40	3	0.4	1604	1444	1456	1684	1614
40	3	0.6	722	802	408	642	722
40	3	0.8	324	404	82	242	322
50	3	0.2	5002	4406	4802	4848	5002
50	3	0.4	3214	2406	2706	2528	3596
50	3	0.6	506	802	804	606	1016
50	3	0.8	102	306	502	302	202

Table 3.12: Number of Nodes in the B&B Tree for 30,40,50-node Examples (continued)

Chapter 4

Heuristic Approaches to Solving SCLAP

In Chapter 3, we have studied the Supply Connected Location-Allocation Problem on networks. We provided lower and upper bounds for the problem and presented a single assignment branch-and-bound algorithm to solve it. The algorithm was then tested on a set of problems and computational results were given. Since the number of branches of the branch-and-bound tree increases exponentially with the number of demand points (n) and the number of facilities to be located (p), the processing times increase rapidly as one of those increases. An important observation is the significance of the role of one of the problem parameters (α) on the computational solvability of the problem: as α varies from 0 to 1, the computation times decrease significantly. This comes from the fact that, as α moves away from 0 closer to 1, the difference between lower and upper bounds decreases allowing more branches to be pruned at the early stages of the branch-and-bound algorithm. However, in most practical cases α is expected to be closer to 0, rather than to 1. The need thus arises for other ways of solving problems in larger sizes.

In accordance with the observation above, the next step to take in this problem is to look for efficient heuristics which might provide reasonable solutions in shorter processing times.

4.1 Local Search Heuristics

Local Search (LS) heuristics are based on searching a small subset of the solution space. For a given point in the solution space, neighboring points within a restricted neighborhood of this point are searched for a new point that is better with respect to some measure. Termination occurs whenever the neighborhood of the current iterate does not contain a better point. Note that the definition of the term "neighborhood" in this chapter differs from the definition of neighborhood in Section 1.2 of Chapter 1, where some existing heuristic algorithms for the location-allocation type problems in the literature were being discussed. Namely, in Section 1.2 the term "neighborhood" referred to the set of demand points (which also constituted the set of candidate facility sites) that were nearest the specific facility location in a feasible solution. In this chapter however, the "neighborhood" of a feasible solution is defined to be the set of feasible solutions that can be derived from the current feasible solution by changing a limited number of items of the solution vector. A more formal definition is given below:

Let Z be the finite set of all possible feasible solutions. For a point $z \in Z$ let $N(z)$ denote the neighborhood of z and for any two points $z^0, z^1 \in Z$ let $d(z^0, z^1)$ denote the distance between points $z^0, z^1 \in Z$. The *distance* is defined to be the function $d : Z \times Z \rightarrow \mathcal{R}$ which satisfies the nonnegativity, triangle inequality, and

symmetry properties. The distance from any point to itself is zero, and this is the only case when the distance is zero. For any parameter $\delta \geq 0$ the δ-neighborhood of $z^0 \in Z$ is defined as $N_\delta(z^0) = \{z^1 \in Z - \{z^0\} : d(z^0, z^1) \leq \delta\}$. For the case of discrete optimization problems, we have to use the following definition (Damberg and Migdalas 1994):

Definition 1: *Let Z be the set of all ordered ν-dimensional vectors of zeros and ones. The Hamming distance on Z is defined by $d_H(z^0, z^1)$= number of positions where z^0 and z^1 are different. The Hamming neighborhood is defined for any positive integer δ by $N_\delta(z^0) = \{z \in Z \ \{z^0\} : d_H(z^0, z^1) \leq \delta\}$*

Note that $z \in N_\delta(z^0)$ if it differs from z^0 in at most δ positions. Hence the size of the neighborhood is $|N_\delta(z^0)| = \sum_{i=1}^{\delta} \binom{\nu}{i}$. Examination of all points in such a neighborhood requires at least $O(\nu^\delta)$ time. Therefore , in practice, one chooses $\delta = 1$ or $\delta = 2$.

For example, in the Location-Routing-Allocation heuristic which we will discuss in the following pages, we have a vector X of size $p \times n$, where each entry of value zero or 1 represents the location assignment of facility j to candidate site i which will be visited at the j-th position on the supply route. At each iteration of the heuristic we assign a facility to another site or change the positions of two facilities on the supply route. Thus, δ is set to 2 in our study.

Two different approaches can be utilized to strengthen the solutions obtained in the LS heuristics. The first is to consider larger neighborhoods at each iteration. In this case computations in each iteration become more complex as the size of the neighborhood increases. Secondly, a more balanced approach can be taken where one would diversify the solution space of the problem's essential variables. That

is, a set of initial points is generated uniformly over the solution space and the LS algorithm is executed from each of these points. Clearly, execution of the algorithm for a large number of starting points uniformly distributed over the solution space increases the probability of finding a global optimum.

LRPs can be described as a combination of three distinct components: (i) *location* of facilities; (ii) *allocation* of demand points to facilities; (iii) *routing*. These components are closely interrelated and should be optimized simultaneously. However, comprehensive mathematical programming formulations which incorporate all aspects of the problem will generally contain too many variables and constraints to be easily solvable. Therefore most of the non-Lagrangian heuristic algorithms used for LRP exploit the decomposition of the problem into its components. Such approaches in general belong to the family of *local search* procedures.

Next we will discuss three different types of heuristics, which we call Location-Routing-Allocation (LRA), Allocation-Routing-Location (ARL), and Location-Allocation-Routing (LAR) heuristics, depending on the relative sequence of decisions. In these heuristics, the basic approach is to change the two decision components with controlled movements within the neighborhood of a given feasible solution and make the third decision optimally with respect to two previous decisions. The procedure begins with a randomly selected starting feasible solution.

Before introducing the heuristics, let us define some of the terminology we use in describing the algorithms. In relation to the three decision components inherent in the problem we have to distinguish between a set of candidate sites (node numbers) for facilities, a sequence (order) of visiting the facilities on the supply route, and an allocation of demand points to facility sites. In this context, in the remaining

part of this chapter, whenever we use the terms *site* or *location*, they will refer to the actual candidate points on which the facility can be built. The term *position* will refer to the specific sequence in the supply tour according to which facility will be visited. With this convention, we then define a *relocation move* which means that a facility is assigned to another candidate site, but the order of visiting the facilities on the supply route does not change. A *pairwise interchange* of facilities means that the positions of two facilities on the supply route will be interchanged, but the set of facility sites which contain an open facility does not change. Similarly, an *allocation* of a demand point to another facility will mean that that demand point is allocated to another facility according to its position on the supply route. Thus the allocation sets change but the visiting sequence on the supply route remains the same. Finally, when we use the term pairwise interchange for allocation sets, it will refer to the pairwise change in the position of facilities to which the two specific allocation sets are assigned; however the allocation sets themselves will not change.

4.2 Location-Routing-Allocation (LRA) Heuristic

The LRA heuristic starts with a set of open facility sites and a predetermined route among them. Observe that in our problem, when the locations of facilities are fixed, determining the supply route among facilities optimally in polynomial time is not possible without knowing the demand point allocations. For that reason, we have to set the sequence of the facilities visited on the supply route beforehand. Once the locations of facilities and their positions on the supply route are known, optimal allocations can be done in polynomial time by utilizing Theorem 3.2 in Chapter 3.

The cost of allocating one unit of demand of a demand point to a given facility at a known position includes two components, say C_1 and C_2. C_1 is the cost of transporting one unit of demand from the facility to the demand point and C_2 is the cost of bringing this demand from the supply plant to the corresponding facility. We assume that the cost of transportation is directly related to the distance. Hence C_1 is the shortest distance between the demand point and the facility and C_2 is the distance on the supply route from supply plant to this facility multiplied by the discount factor α. Since the locations and positions of all facilities are fixed, C_1 and C_2 are easily computable for each demand point. The sum of C_1 and C_2 gives the unit cost of transportation starting from the supply plant to the given demand point. Thus each demand point is allocated to a facility on the supply route which gives the least unit transportation cost. This can be done in $O(p)$ time for each demand point by evaluating all facilities. Hence, determining the best allocations for all demand points can be done in $O(np)$ time.

Now let us summarize the LRA algorithm. The LRA algorithm starts with an arbitrary set of p facility sites and a preset order of visiting among them. Since the locations and positions of the facilities on the supply route are given, the demand points are then allocated to the facilities according to Theorem 3.2. At each iteration of the search we change the location and routing components of the problem and modify the allocation component accordingly. In order to utilize a larger neighborhood, we incorporate a nested loop of two repetitive actions. That is, instead of making relocation and interchange steps independently, we first make a relocation move, and then we proceed with the pairwise interchange of facility positions. This increases the size of the search neighborhood at each step. If all possible pairwise

interchange of facility positions do not give any reduction in the total transportation cost, we evaluate the first relocation move which may still provide improvements in the objective function. As soon as an improvement is reached, the best objective function value so far is updated, and the new configuration of facility locations and positions, together with the consequent allocations, is taken as the new starting point and the procedure returns to the beginning. The algorithm continues until all possible relocations and all possible interchanges within each relocation move of a given configuration have been considered and no further improvements are possible.

We call this algorithm LRA(A) because we also study a slightly modified version of the LRA heuristic in this chapter, which we call LRA(B). The steps of the LRA(A) algorithm are given next, and Figure 4.1 shows the flowchart of the algorithm.

Algorithm LRA(A)

0. Select an arbitrary set of p facility sites and assign a sequence among them.

1. *If* all facilities in this configuration have been considered for relocation,
   then go to STEP 5,
   else select a facility to relocate on an available site.

2. *If* all possible pairs of facilities have been considered for position interchange on the supply route
   then go to STEP 4,
   else select the next pair of facilities to interchange their positions.

3. *If* the interchange move improves the objective function (i.e. reduces cost), *then* select this solution as the next starting point and return to STEP 1, *else* return to STEP 2.

4. *If* the relocation move improves the objective function value, *then* select this solution as the next starting point and return to STEP 1.

5. Keep the best solution so far and **STOP**.

Steps 2 and 3 of the algorithm correspond to the inner loop of the *interchange* operation and steps 1 and 4 correspond to the outer loop of the *relocation* operation.

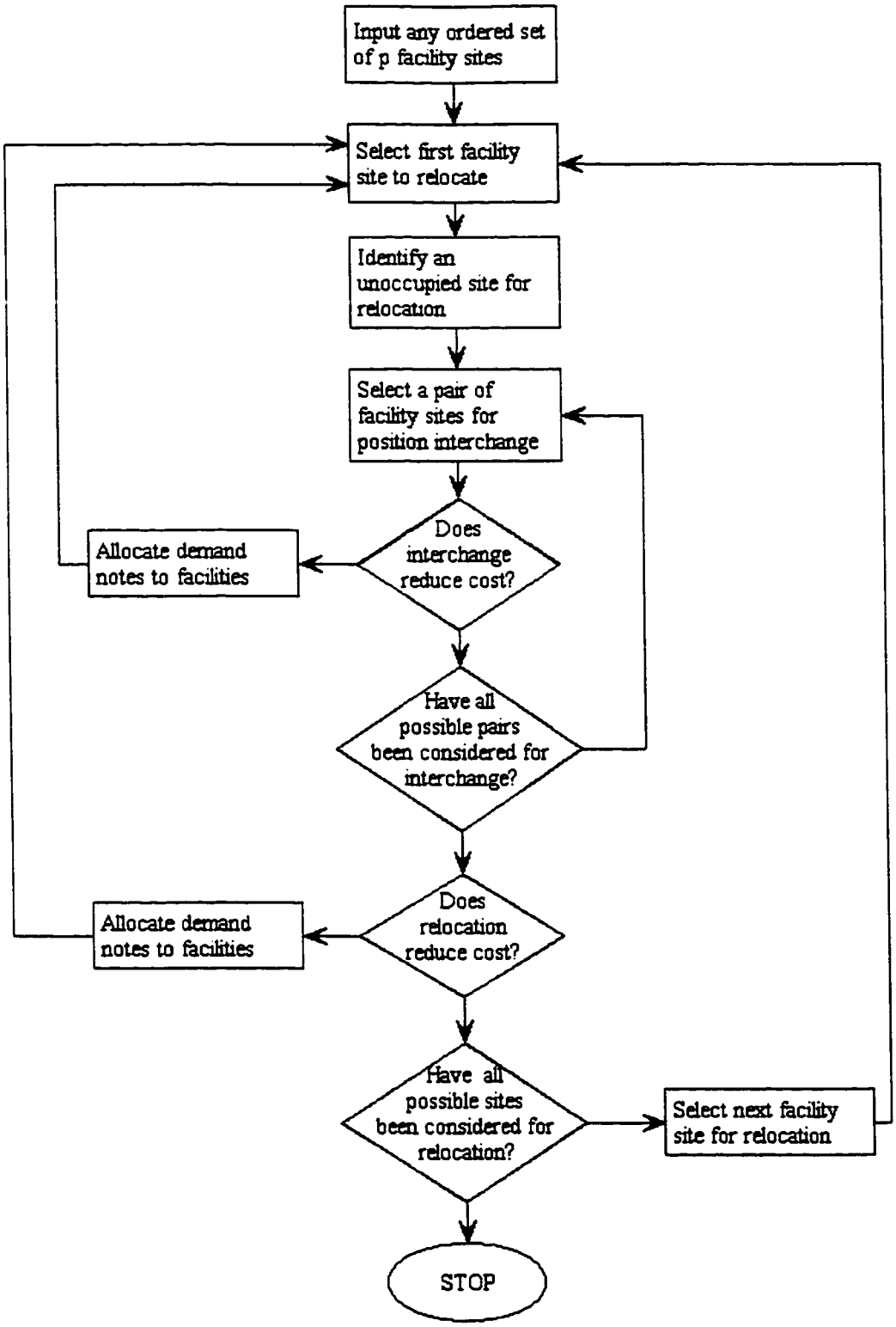

Figure 4.1: Flowchart of Algorithm LRA(A)

We also study a slightly modified version of the LRA heuristic, called LRA(B) as mentioned earlier. In this version, instead of selecting the first improvement point as the next starting configuration, we search all possible relocation movements (i.e. the entire neighborhood with respect to relocation moves), as well as all possible interchange movements within each relocation (i.e. for every solution in the neighborhood of the starting point with respect to relocation, we search the entire neighborhood of each solution with respect to position interchange) and select the one that results in the greatest improvement in the objective function. Thus the steps 3 and 4 of the algorithm are changed to:

3'. *If* the interchange move improves the objective function (i.e. reduces cost), *then* store this solution as the current best solution.
Return to STEP 2.

4'. *If* the relocation move improves the objective function value, *then* store this solution as the current best solution
Return to STEP 1.

In the algorithm LRA(B), we essentially search for the local minimum of the neighborhood of the given starting point and select that local minimum as the next starting point.

4.3 Allocation-Routing-Location (ARL) Heuristic

The second heuristic we study is called the Allocation-Routing-Location (ARL) heuristic. The idea now is to update the allocation and routing decisions with moves in the local search neighborhood and locate the facilities with respect to the given

allocation and routing configuration. For that we partition the demand set into p subsets and assign a sequence to the given p subsets. That is, each subset is given a unique position which determines the sequence by which the facility that serves the demand points in this subset will be visited on the supply route. Once we know the allocation sets and their relative sequence on the supply route, we can solve for the actual locations of facilities by a backward dynamic programming procedure, essentially similar to the one described for the case on a line described in Chapter 2. For algorithmic purposes note that when we want to locate, say, k-th facility, $(k-1)$st and $(k+1)$st facilities behave like demand points to facility k with weights $\left(\alpha \sum_{i \in \cup_{l=k}^{p} I_l} w_i\right)$ and $\left(\alpha \sum_{i \in \cup_{l=k+1}^{p} I_l} w_i\right)$ respectively, where I_l denotes the set of indices of demand points which are allocated to the l-th facility. The transportation cost between the $(k-1)$st, k-th, and $(k+1)$st facilities on the supply route is given by:

$$\left(\alpha \sum_{i \in \cup_{l=k}^{p} I_l} w_i\right) d(x_{k-1}, x_k) + \left(\alpha \sum_{i \in \cup_{l=k+1}^{p} I_l} w_i\right) d(x_k, x_{k+1}). \tag{4.1}$$

To solve for the locations via dynamic programming we first consider each candidate site for the $(p-1)$st facility and find the location of the p-th facility with respect to the $(p-1)$st facility's given location. Then, for each candidate site of the $(p-2)$nd facility, we find the location of the $(p-1)$st facility and so on. The dynamic programming relation is given by

$$f_k^*(x_{k-1}, x_k^*) = \min_{x_k \in V} \left\{ \sum_{i \in I_k} w_i d(x_k, v_i) + \left(\alpha \sum_{i \in \cup_{l=k}^{p} I_l} w_i\right) d(v, x_k) + f_{k+1}^*(x_k, x_{k+1}^*) \right\}. \tag{4.2}$$

where $f_k^*(v, x_k^*)$ denotes the minimum transportation cost from a given point v to the next facility k on the supply route and the cost of allocating demand points to the

k-th, $(k+1)$st,...,p-th facilities, as well as the respective transportation costs between those facilities on the supply route. V_k denotes the set of demand points that are allocated to the k-th facility, and x_k denotes the location of the k-th facility.

We start with the location of p-th facility for each candidate site, v, for the $(p-1)$st facility. Then we solve:

$$f_p^*(v, x_p^*) = min_{x_p \in V} \left\{ \sum_{i \in I_p} w_i d(x_p, v_i) + \left(\alpha \sum_{i \in I_p} w_i \right) d(v, x_p) \right\}. \qquad (4.3)$$

Next we solve for the $(p-1)$st, the $(p-2)$nd facility and so on, using the recursive relation given (in 4.2). When we reach the location of first facility, we have only one candidate site for the 0-th facility, so to speak, which is actually the supply plant itself. Note that earlier in the thesis, we indicated that the supply plant is located at a known point and for notational simplicity, we decided that this location would be indicated by v_0. Thus the final step in the dynamic programming procedure will consist of solving

$$f_1^*(v_0, x_1^*) = min_{x_1 \in V} \left\{ \sum_{i \in I_1} w_i d(x_k, v_i) + \left(\alpha \sum_{i \in I} w_i \right) d(v_0, x_1) + f_2^*(x_1, x_2^*) \right\}. \quad (4.4)$$

where $I = \cup_{l=1}^{p} I_l$.

(4.4) gives the minimum transportation cost for the given allocation and routing configuration. Once we solve (4.4) we can trace back the optimal locations, $x_1^*, x_2^*, \ldots, x_p^*$. At each iteration of the DP algorithm we consider at most n available sites for the given facility and compare the costs for each site. Computing the cost for a given site can be done in $O(n)$ time. Therefore at each iteration (4.2) can be calculated in $O(n^2)$ time. Since the DP algorithm consists of p iterations, each corresponding to location of a facility, the entire DP algorithm will take $O(pn^2)$ time.

Having established the procedure for locating the facilities for the given allocation and routing configuration in the ARL heuristic, we developed a search algorithm similar to the one described in the previous section. For the case of the ARL heuristic we also consider two slightly different versions as we did in the LRA heuristic. Namely, in the first case we restart the algorithm as soon as a solution that improves the objective function value is reached, and in the second case, we search for the best possible improvement in the neighborhood of the given starting solution and select the one that gives the best improvement as the next starting point.

Algorithm ARL(A) starts with a given partition of demand points into p subsets and a preset sequence among them. Given the allocation sets and positions for the facilities, the locations of facilities are then calculated using the backward dynamic programming procedure described above. At each iteration of the algorithm ARL(A) we go to a neighboring point of the given solution with respect to the allocation and routing changes. In order to accommodate a larger search neighborhood, we first identify a pair of allocation sets to interchange their positions and then for the given interchange move, we consider reallocating demand points one at a time to a facility other than the current facility they are allocated to. If any reallocation move results in an improvement of the objective function (i.e. reduces cost) we select the given configuration (together with the demand reallocation and facility position interchange, as well as the consequent location changes) as the next starting point and restart the algorithm, otherwise we consider another demand point to reallocate until all demand points are considered. If none of the reallocation moves results in an improvement in the objective function value, we consider the initial interchange move which may still result in a decrease in cost. If an improvement is possible, the given

configuration is selected as the next starting point and the algorithm is restarted; otherwise another pair of allocation sets is considered for position interchange, until all possible pairwise interchanges are considered. The algorithm continues in this manner until no further improvements are possible. The steps of the algorithm ARL(A) is given next and Figure 4.2 shows the flowchart of the algorithm.

Algorithm ARL(A)

0. Select an arbitrary partition of demand points into p subsets and assign a sequence among them.

1. *If* all possible pairs of allocation sets in this configuration have been considered for position interchange,
then go to STEP 5.
else select a pair of allocation sets to interchange their facilities' positions on the supply route.

2. *If* all demand points have been considered for reallocation to another facility
then go to STEP 4,
else select the next demand point to reallocate.

3. *If* the reallocation move improves the objective function (i.e. reduces cost),
then select this solution as the next starting point and return to STEP 1,
else return to STEP 2.

4. *If* the interchange move improves the objective function value,
then select this solution as the next starting point and return to STEP 1.

5. Keep the best solution so far and **STOP**.

Steps 2 and 3 of the algorithm correspond to the *reallocation* operation and steps 1 and 4 correspond to the *position interchange* operation.

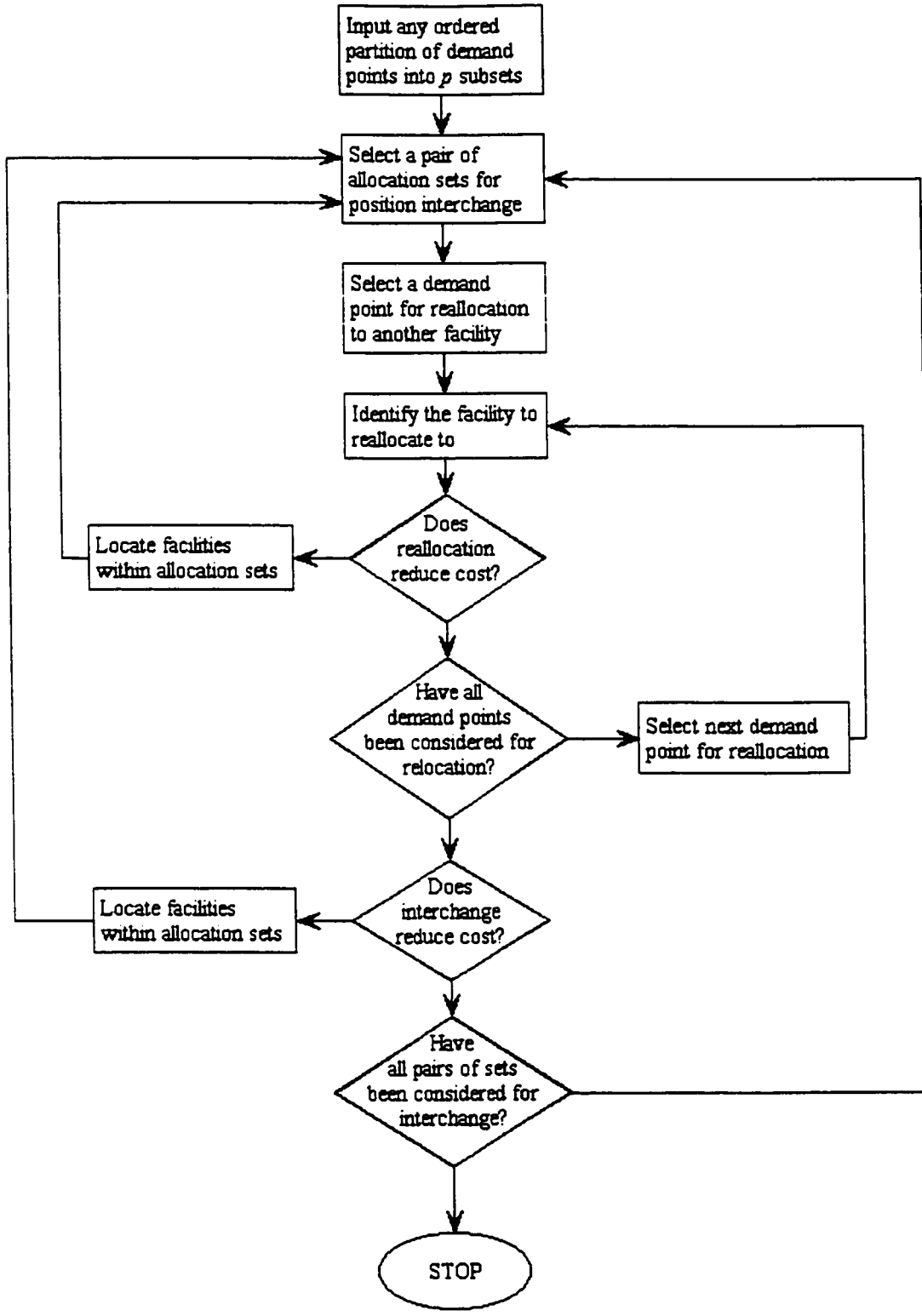

Figure 4.2: Flowchart of Algorithm ARL(A)

As we mentioned earlier, we also study a slightly modified version of the algorithm ARL(A), which we call ARL(B). In this version we search all possible reallocation and position interchange operations for best improvement in the objective function, before we select the next starting point. That is, we select the local minimum of the given neighborhood as the next starting point. Thus the steps 3 and 4 of the algorithm are changed to:

3″. *If* the reallocation move improves the objective function (i.e. reduces cost), *then* store this solution as the current best solution.
Return to STEP 2.

4″. *If* the interchange move improves the objective function value, *then* store this solution as the current best solution
Return to STEP 1.

4.4 Location-Allocation-Routing (LAR) Heuristic

The idea behind the heuristics described in this chapter is simple; namely, we change the two decision components of the problem with movements in the neighborhood of the starting solution and solve the third decision component optimally with respect to the given two decision components. For example, in the LRA heuristic, we preset the location and routing decision by movements in the single relocation or repositioning movements in the neighborhood of each starting point and solved the allocation decision with respect to the given location and routing decisions optimally. Similarly in the ARL heuristic allocation and routing decisions were set by movements in the neighborhood of the starting solution and the locations were found optimally with

respect to the given allocation and routing decisions.

This approach clearly suggests the idea of a third type of heuristic which could be called Location-Allocation-Routing (LAR) heuristic, where one would determine the location and allocation decisions by simple movements in the neighborhood of the given feasible solution and solve the routing decision accordingly. Although the idea is simple, there is an interesting complication from the viewpoint of computational cost. Recall from the analysis in Chapter 1 that when the locations of facilities and allocations of demand points to facilities are known, the resulting problem, which corresponds to the routing decision component, is the Weighted Delivery Man Problem (WDMP) which is NP-Hard. Additionally, no known polynomial time algorithm exists to solve one of its special cases, the DMP, even on tree networks. Thus the problem we have to solve at each iteration of the LAR heuristic, is quite difficult to manage for large values of p (i.e. the number of facilities) in reasonable computation time. For small values of p, this approach may still be useful by solving the subproblem optimally with brute force approach. For example when p is two there are only two possible route sequence and when p is three there are six possible route configurations. Of course this number grows very rapidly with the order of $p!$ as p increases. Because of the computational difficulty mentioned we did not provide computational testing and comparison for this heuristic in this thesis.

As pointed out earlier, these heuristics preset the values for two decision components and then optimize the third component accordingly. Clearly one would ask if it was possible to set decisions for only one component of the problem and then try to optimize the remaining two components simultaneously. Considering the

computational difficulty of even the subproblems, this approach may not seem very promising. However, for very small values of p it may be possible to find the route optimally by complete enumeration. A slightly modified version of ARL heuristic can be obtained by predetermining the allocation decisions only, and then for each allocation group, searching all possible route configurations and finding the optimal locations.

4.5 Computational Testing and Performance Comparison of Heuristics

The four algorithms discussed are coded in C++ and tested on a total of 50 randomly generated networks. The networks considered consisted of 11,21,...,51 nodes, one of which represented the supply plant location and the others represented the demand points as well as the candidate sites for facility locations; 10 different networks are generated for each case. The edge lengths are drawn from a uniform distribution between 1 and 100. The distances are calculated to be the length of the shortest path between any two points on the network. Therefore, if an edge length does not conform to triangle inequality property of a distance metric so defined, i.e. if the length of the edge is greater than the length of the shortest path between its endpoints, then the solutions simply would not include that edge. This is because the transportation costs are calculated according to distances defined by the shortest path lengths, which means any shipment not using the shortest path between two points would cost more. The problems are then solved for different p (number of facilities to locate) and α (scaling factor) values on each of the randomly generated networks. To further improve the results of each heuristic, we ran all the heuristic

algorithms from 10 randomly generated starting points and selected the best solution among them. For a more reliable comparison and to avoid effects of the starting point selection, the two different (A and B) versions of each type of heuristic were started from the same starting points. All the algorithms were run on a Sun Sparc V with 70 mHz processor.

The results of the heuristics and their computation times are compared with the optimal results obtained by the B&B algorithm described in Chapter 3, whenever optimal results were available. Tables 4.1 through 4.6 compare the performance of heuristics with respect to the B&B Algorithm and the resulting optimal solutions. Tables 4.1, 4.2, and 4.3 give the average processing times of all algorithms over 10 test problems for each n, p, α combination. Tables 4.4, 4.5, and 4.6 give the average results on percentage error (i.e. how far the heuristic solution is away from the optimum). Tables 4.7, 4.8, and 4.9 compares the results of heuristics between themselves for larger instances of the problem, where no optimal solutions were available.

From Tables 4.1, 4.2, and 4.3 we see that for small instances of the problem where n (number of facilities) and p (number of demand points) are relatively small, B&B algorithm performs better in terms of computation time. However, of course, as the problem size increases the time to solve each problem increases very rapidly and falls behind the heuristic algorithms. Increase in computation time for all algorithms both exact and heuristic is much more sensitive to an increase in p than in n. The change in scaling factor α does not seem to affect the computation times of heuristics as significantly as it does for B&B algorithm. Of course, the importance of α was expected in the B&B algorithm because of the difference α causes between lower and

n	p	α	Average CPU (in seconds)				
			B&B	LRA(a)	LRA(b)	ARL(a)	ARL(b)
10	3	0.1	0.1	0.0	0.2	0.7	0.5
10	3	0.2	0.0	0.2	0.1	0.2	0.5
10	3	0.3	0.0	0.0	0.1	0.4	0.5
10	3	0.4	0.0	0.2	0.2	0.5	0.3
10	3	0.5	0.0	0.1	0.0	0.2	0.6
10	3	0.6	0.0	0.0	0.1	0.2	0.4
10	3	0.7	0.0	0.0	0.2	0.4	0.2
10	3	0.8	0.0	0.1	0.0	0.3	0.5
10	3	0.9	0.0	0.2	0.0	0.0	0.3
10	5	0.1	2.0	0.5	0.3	4.8	7.5
10	5	0.2	1.5	0.3	0.6	5.2	7.2
10	5	0.3	1.1	0.4	0.2	5.2	6.3
10	5	0.4	1.0	0.5	0.8	6.0	6.3
10	5	0.5	0.8	0.2	0.2	5.2	6.6
10	5	0.6	0.6	0.4	0.4	5.3	5.5
10	5	0.7	0.4	0.3	0.5	5.5	5.8
10	5	0.8	0.4	0.5	0.4	5.2	6.0
10	5	0.9	0.3	0.1	0.6	5.3	6.1

Table 4.1: CPU for 10 node examples

			Average CPU (in seconds)				
n	**p**	α	**B&B**	**LRA(a)**	**LRA(b)**	**ARL(a)**	**ARL(b)**
20	3	0.1	0.4	0.6	0.2	4.2	6.7
20	3	0.2	0.8	0.3	0.9	3.6	5.3
20	3	0.3	0.5	0.7	0.1	3.4	4.5
20	3	0.4	0.4	0.3	0.8	2.6	4.1
20	3	0.5	0.3	0.7	0.2	2.9	3.6
20	3	0.6	0.2	0.3	0.7	2.6	3.5
20	3	0.7	0.1	0.6	0.5	1.9	3.2
20	3	0.8	0.3	0.5	0.3	2.2	2.8
20	3	0.9	0.0	0.2	0.5	2.7	3.3
20	5	0.1	190.4	3.1	3.2	38.7	73.4
20	5	0.2	148.9	3.3	3.0	42.0	59.0
20	5	0.3	112.2	3.3	2.9	45.3	53.2
20	5	0.4	91.5	2.8	2.5	51.2	47.8
20	5	0.5	72.3	2.3	2.3	49.0	43.6
20	5	0.6	50.0	3.0	2.5	48.4	43.8
20	5	0.7	36.0	2.5	2.4	47.6	38.4
20	5	0.8	26.8	2.7	2.4	45.6	41.3
20	5	0.9	14.5	2.2	2.4	43.2	44.7

Table 4.2: CPU for 20 node examples

n	p	α	Average CPU (in seconds)				
			B&B	LRA(a)	LRA(b)	ARL(a)	ARL(b)
30	3	0.2	2.9	1.3	1.0	16.0	22.9
30	3	0.4	1.3	1.3	1.2	10.6	14.5
30	3	0.6	0.8	1.2	1.2	10.1	11.9
30	3	0.8	0.2	1.6	1.1	11.9	11.4
30	5	0.2	2223.1	11.5	9.2	253.1	277.3
30	5	0.4	1150.1	9.1	8.1	292.6	202.0
30	5	0.6	507.9	8.6	6.4	261.5	154.8
30	5	0.8	229.1	7.9	6.7	236.9	179.2
40	3	0.2	9.2	2.7	6.0	47.8	52.4
40	3	0.4	4.5	2.6	5.8	32.0	38.0
40	3	0.6	1.9	3.1	4.5	21.4	28.5
40	3	0.8	0.6	3.0	4.3	20.7	28.3
50	3	0.2	22.7	5.4	3.7	120.7	113.3
50	3	0.4	12.4	5.3	4.0	84.3	86.2
50	3	0.6	3.6	5.3	3.0	54.5	67.3
50	3	0.8	1.4	5.0	3.1	55.0	62.6

Table 4.3: CPU for 30, 40 50 node examples

upper bounds used at each iteration of the algorithm, as discussed in Chapter 3. One important observation from processing time tables is that both LRA type heuristics are much faster than both ARL type heuristics. This was expected because each iteration of a LRA heuristic requires $O(np)$ steps, whereas each iteration of an ARL heuristic requires $O(n^2p)$ steps. Thus, on average, one might expect an ARL type heuristic to be about $O(n)$ times slower than the LRA type heuristics, this is shown in Tables 4.1, 4.2, and 4.3.

			% Error			
n	p	α	LRA(a)	LRA(b)	ARL(a)	ARL(b)
10	3	0.1	0.000	1.246	2.589	0.201
10	3	0.2	0.000	0.064	2.580	1.972
10	3	0.3	0.165	0.219	4.780	5.526
10	3	0.4	0.000	0.901	4.924	1.992
10	3	0.5	0.000	0.276	5.277	3.426
10	3	0.6	0.000	0.837	5.499	3.268
10	3	0.7	0.000	0.234	4.180	1.902
10	3	0.8	0.000	0.152	2.240	1.448
10	3	0.9	0.000	0.000	1.087	1.182
10	5	0.1	0.113	0.000	0.129	0.375
10	5	0.2	0.000	0.000	2.418	0.444
10	5	0.3	0.474	0.151	1.399	0.157
10	5	0.4	0.322	0.000	1.655	1.541
10	5	0.5	0.004	0.090	3.163	1.949
10	5	0.6	0.003	0.004	2.919	2.438
10	5	0.7	0.006	0.006	3.784	2.706
10	5	0.8	0.004	0.003	3.326	3.087
10	5	0.9	0.002	0.002	3.069	2.920

Table 4.4: % Error for 10 node examples

Tables 4.4, 4.5, and 4.6 show, on average, how far the best solution found by each heuristic was away from the optimum solution. The percentage error for each

n	p	α	% Error			
			LRA(a)	LRA(b)	ARL(a)	ARL(b)
20	3	0.1	0.000	0.052	2.591	2.255
20	3	0.2	0.000	0.688	5.841	4.302
20	3	0.3	0.052	0.626	5.073	4.496
20	3	0.4	0.000	0.188	7.592	6.491
20	3	0.5	0.000	0.543	5.790	5.571
20	3	0.6	0.000	0.446	6.374	5.538
20	3	0.7	0.000	0.347	6.721	5.178
20	3	0.8	0.000	0.237	4.682	3.958
20	3	0.9	0.009	0.201	2.132	2.032
20	5	0.1	0.014	0.101	1.637	2.543
20	5	0.2	0.047	0.022	3.364	2.858
20	5	0.3	0.202	0.356	4.890	3.111
20	5	0.4	0.590	0.295	4.114	3.164
20	5	0.5	0.235	0.101	3.065	3.634
20	5	0.6	0.053	0.053	3.682	2.693
20	5	0.7	0.119	0.153	2.533	2.785
20	5	0.8	0.123	0.107	2.713	1.633
20	5	0.9	0.046	0.035	1.164	0.889

Table 4.5: % Error for 20 node examples

n	p	α	% Error			
			LRA(a)	LRA(b)	ARL(a)	ARL(b)
30	3	0.2	0.131	0.470	7.214	10.795
30	3	0.4	0.022	0.484	9.451	8.437
30	3	0.6	0.000	0.602	7.226	6.468
30	3	0.8	0.002	0.280	3.559	3.537
30	5	0.2	0.000	0.151	7.194	8.817
30	5	0.4	0.231	0.583	5.777	8.334
30	5	0.6	0.350	0.228	3.873	6.062
30	5	0.8	0.116	0.123	2.874	3.261
40	3	0.2	0.000	0.643	9.643	12.104
40	3	0.4	0.110	0.911	8.597	8.911
40	3	0.6	0.000	0.264	9.526	8.771
40	3	0.8	0.005	0.342	5.304	5.665
50	3	0.2	0.000	0.276	12.008	13.561
50	3	0.4	0.129	0.592	10.617	11.495
50	3	0.6	0.000	0.674	8.354	9.114
50	3	0.8	0.000	0.221	4.883	4.739

Table 4.6: % Error for 30, 40 and 50 node examples

test problem was calculated by taking the ratio of the difference between best solution found by the heuristic and the optimum solution found by the B&B algorithm, with the objective value of the optimum. Note that 0.000 means that the algorithm was able to find the optimum solution in all 10 test problems. % Error tables show a clear dominance of the LRA type heuristics over ARL type heuristics in closeness to optimality. In all cases, the LRA type heuristics found solutions which were close to the optimum within 1% range on average, whereas ARL type heuristics would find solutions up to about 12% closeness. For the problems we study in this group (i.e. smaller problems where optimum solutions were available), interestingly, it seems that heuristic LRA(A) performs better than the heuristic LRA(B), suggesting that searching for the best solution in the neighborhood of a given solution may lead to algorithm being stuck in a "bad" local minimum more easily. However, in the case of ARL type heuristics ARL(B) performed better relative to ARL(A) for small problems. However for larger problems ARL(A) outperformed ARL(B). The computation times for ARL(B) were less than the computation times for ARL(A) for large problem sizes. This may imply that the ARL(B) gets stuck in a "bad" neighborhood resulting in a worse local minimum more easily.

We terminated the B&B algorithm for instances where the computation time for a problem exceeded 1 hour of CPU. Therefore, we did not have optimal results for larger problem sizes. However, we continued the four heuristics for some larger problems and compared their performance with respect to each other. Tables 4.7, 4.8, and 4.9 provide the results for those instances. In this case percentage error was calculated relative to the best solution found among all algorithms. Therefore we also included the information of how many times the best solution was found by each

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	13.0	10.9	239.8	350.6
20	7	0.2	13.4	9.8	291.9	308.4
20	7	0.3	10.6	8.3	258.7	273.8
20	7	0.4	7.2	7.6	298.1	275.5
20	7	0.5	7.0	7.2	292.8	248.0
20	7	0.6	8.5	7.6	271.3	240.9
20	7	0.7	6.9	7.2	329.5	258.5
20	7	0.8	6.2	7.5	300.3	259.2
20	7	0.9	5.9	7.8	300.2	253.5
30	7	0.2	43.7	27.0	1426.4	1336.8
30	7	0.4	28.0	21.9	1612.9	1098.3
30	7	0.6	23.6	21.7	1450.9	963.6
30	7	0.8	20.2	20.9	1349.5	946.7
40	5	0.2	25.7	38.3	1029.6	818.6
40	5	0.4	23.8	31.8	908.3	541.1
40	5	0.6	19.3	31.1	1033.8	538.0
40	5	0.8	19.7	33.6	783.6	502.9
50	5	0.2	48.3	26.2	2442.0	1901.6
50	5	0.4	49.5	23.2	2237.0	1334.5
50	5	0.6	39.8	21.4	2161.1	1075.4
50	5	0.8	36.0	20.2	2094.0	1051.7

Table 4.7: Average Processing Times of the Heuristic Algorithms

n	p	α	Average % Error			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	0.116	0.036	2.235	2.149
20	7	0.2	0.565	0.089	2.387	2.876
20	7	0.3	0.412	0.127	3.102	3.538
20	7	0.4	0.495	0.257	2.120	1.474
20	7	0.5	0.424	0.057	3.376	4.177
20	7	0.6	0.060	0.040	3.030	3.896
20	7	0.7	0.126	0.035	3.073	3.013
20	7	0.8	0.116	0.025	3.714	4.060
20	7	0.9	0.023	0.000	2.693	3.752
30	7	0.2	0.358	0.029	3.784	4.978
30	7	0.4	0.453	0.033	4.336	4.920
30	7	0.6	0.333	0.000	3.479	3.251
30	7	0.8	0.124	0.000	3.172	3.424
40	5	0.2	0.520	0.117	5.823	5.943
40	5	0.4	0.291	0.137	5.511	7.871
40	5	0.6	0.004	0.049	3.912	5.910
40	5	0.8	0.015	0.052	3.869	3.938
50	5	0.2	0.074	0.333	6.144	8.155
50	5	0.4	0.000	0.172	6.040	7.392
50	5	0.6	0.028	0.061	4.684	6.046
50	5	0.8	0.013	0.013	3.296	3.959

Table 4.8: Average % Difference from the Best Solution Found

n	p	α	Times the Best Found			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	9	9	0	0
20	7	0.2	8	7	2	0
20	7	0.3	7	7	0	0
20	7	0.4	4	7	0	3
20	7	0.5	5	8	0	0
20	7	0.6	8	9	0	0
20	7	0.7	6	9	0	1
20	7	0.8	6	9	0	1
20	7	0.9	8	10	0	0
30	7	0.2	6	7	0	0
30	7	0.4	6	9	0	0
30	7	0.6	5	10	0	0
30	7	0.8	7	10	0	0
40	5	0.2	5	8	1	0
40	5	0.4	9	7	0	0
40	5	0.6	9	7	0	0
40	5	0.8	9	9	0	0
50	5	0.2	8	5	0	0
50	5	0.4	10	4	0	0
50	5	0.6	8	8	0	0
50	5	0.8	9	7	0	0

Table 4.9: Number of Times Best Solution was Found by Each Heuristic

algorithm over 10 test problems for each n, p, and α combination. Results discussed earlier for ARL type algorithms seemed to hold the same here as well. What is interesting to note is that algorithm LRA(B) started performing better than LRA(A) for large problems, it found the best solution more times than LRA(A).

To have a better comparison for the LRA type heuristics between themselves we tried even larger problem instances for these LRA(A) and LRA(B) heuristics, the results of which are presented in Table 4.10. As the problem sizes increased with respect to number of demand points and number of facilities to locate, the algorithm LRA(B) found a better solution more times than the LRA(A). What is also important to note is that the computation times were not significantly longer than the LRA(A) algorithm. LRA(B) Algorithm had a relatively uniform computation time with respect to changes in the value of α, whereas LRA(A) seemed to take slightly less time as α moved away from 0 closer to 1. Since LRA(B) searches the entire neighborhood at each iterate, potential effect of α in its computation time is less noticeable.

Finally we also present comparison of all those algorithms for each of the 10 problems for all n, p, and α combinations, individually in Appendices B, C, and D.

From the analysis in this chapter, LRA type heuristics come out as a winner both in terms of computation times and closeness to optimality or best solution. Although LRA(A) seemed to perform better for small problem sizes, it later became clear that LRA(B) gives better solutions for large problem sizes. However, more testing has to be done with larger number of examples for each parameter combination to be able to indicate whether there is a significant statistical difference

			Avg. CPU		% Error		#. Best Find	
n	p	α	LRA(A)	LRA(B)	LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	129.9	83.6	0.172	0.362	6	8
30	10	0.4	69.1	73.9	0.867	0.058	4	9
30	10	0.6	59.9	74.6	0.151	0.000	6	10
30	10	0.8	54.1	72.5	0.099	0.000	7	10
40	7	0.2	106.9	138.4	0.241	0.255	7	6
40	7	0.4	86	112	0.547	0.082	8	5
40	7	0.6	50.7	97	0.071	0.000	9	10
40	7	0.8	53.8	93.5	0.064	0.000	9	10
40	10	0.2	281.9	239.7	0.720	0.129	6	7
40	10	0.4	227.1	247.3	0.018	0.021	8	7
40	10	0.6	130.1	207.8	0.033	0.007	9	8
40	10	0.8	113.9	187.3	0.090	0.000	8	10
40	12	0.2	639.5	765.9	1.088	0.000	5	10
40	12	0.4	349.6	717.9	0.555	0.000	4	10
40	12	0.6	213.8	677.2	0.008	0.057	8	9
40	12	0.8	253.2	701.2	0.082	0.000	8	10
50	7	0.2	217.3	94.7	0.128	0.187	9	5
50	7	0.4	171	90.5	0.071	0.083	8	6
50	7	0.6	121.3	74.3	0.136	0.037	7	7
50	7	0.8	98.5	66.6	0.026	0.005	9	9
50	10	0.2	762.8	319.6	0.639	0.191	3	7
50	10	0.4	458	266.9	0.343	0.000	3	10
50	10	0.6	346	266.4	0.032	0.168	8	6
50	10	0.8	268.9	245.4	0.000	0.000	10	10
50	12	0.2	1191.7	545.5	0.805	0.134	4	8
50	12	0.4	774.2	519.1	0.110	0.141	7	7
50	12	0.6	552.4	496.6	0.169	0.040	8	8
50	12	0.8	459.8	465.1	0.035	0.000	8	10
50	15	0.2	2056.7	1151.7	0.992	0.169	1	9
50	15	0.4	1429.6	1227.4	0.271	0.045	3	9
50	15	0.6	1036.1	1119.7	0.041	0.041	9	7
50	15	0.8	852.6	1017.4	0.015	0.000	9	10

Table 4.10: Performance of Two Versions of the LRA Type Heuristic for Larger Instances

between performances of LRA(A) and LRA(B), because in general the solutions found by each algorithm were very close to each other. In most cases the difference was less than 2% (see Appendix D) for larger problem sizes.

Chapter 5

Conclusions and Future Research

This chapter presents a summary of the research done in this study and discusses some avenues for future research related to the topics investigated.

5.1 Summary

This research presents a new model that has not been previously studied in the literature. The problem is first investigated for special cases where it is efficiently solvable. Next, an exact solution procedure is developed to solve the problem in its general form. The third approach is to develop heuristic algorithms to construct "good" solutions for large problem instances, although they may not be optimal.

Chapter 1 presents a brief introduction to Location-Routing Problems and discusses the two major components of LRPs, namely the class of Location (or Location-Allocation) and the class of Routing problems, in detail. Certain problems within each class are given special attention, because they appear as subproblems of the SCLAP or they are closely related to variations of the problem. Those problems are the p-median and the uncapacitated facility location problems within the class of location problems and Traveling Salesman and Delivery Man Problems within the

class of routing problems. One of the subproblems of the SCLAP is also identified to be a new model that has not been previously studied in the literature. It is called the Weighted Delivery Man Problem and is closely related to the DMP. The WDMP is a variation of the DMP, where each city in the tour has a positive weight associated with it, and the problem is to minimize the sum of *weighted* distances traveled on the tour from starting point up to each city (in the case where it is assumed that the travel time is directly proportional to the distance traveled, the objective becomes minimizing the sum of weighted waiting times). Later in Appendix A, this problem is studied briefly and a lower bound is proposed on tree networks which could be utilized in a branch-and-bound algorithm to solve the problem. Finally, the problem that is discussed in this study, SCLAP, is introduced.

In Chapter 2, we study the problem on a line and present an efficient (i.e. of polynomial time complexity) dynamic programming solution procedure to solve it. The range of the discounting factor α, for which a given solution remains optimal, is also determined. Next the problem is studied on tree networks and it is shown that the dynamic programming procedure can be extended to solve it for a special case where the supply route is constrained to consist of a single path.

In Chapter 3, we develop a single assignment branch-and-bound procedure to solve the SCLAP on general networks. In single assignment, a new facility is opened at each step on one of the available candidate sites. Then the lower and upper bounds on the optimal objective function value involving this assignment and previously located facilities are calculated. These bounds are then used in a branch-and-bound algorithm to solve the problem optimally. The branching rule used is to select the available facility site which gives the lowest upper bound. The algorithm

is then tested on randomly generated networks and the results with respect to the computation time and the size of the branch-and-bound tree are presented. As one would expect with nonpolynomial algorithms, the computation time and the size of the branch and bound tree increased very rapidly as the size of the problem instances increased with the number of demand points (n) and number of facilities to locate (p). However, development of this type of algorithm is still justified, first because it is more efficient than a brute force approach and lets relatively larger instances of problem to be solved to optimality. More importantly, this algorithm provides a concrete basis for comparing performances of heuristic algorithms and help us predict their performance for larger problem instances.

In Chapter 4, we discuss three local search type heuristic approaches for SCLAP. These heuristics are based on searching the neighborhood of a given starting feasible solution, for improvement in the objective function. In each of these heuristics, two of the decision components of the SCLAP are changed through movements within the local neighborhood and the third decision component is calculated optimally with respect to the two decisions made in this way. For example, in the Location-Routing-Allocation (LRA) heuristic, the location and routing decisions are changed by relocating a facility to another candidate site or interchanging the positions of two facilities in the visiting sequence of the supply route. The allocations are then calculated optimally (in O(np) time) with respect to changes in the location and routing decisions. Similarly, in the Allocation-Routing-Location (ARL) heuristic, allocation and routing decisions are updated through movements within the neighborhood of the given feasible solution and the location component is calculated optimally via a dynamic programming solution procedure in $O(pn^2)$ time.

Finally, the Location-Allocation-Routing (LAR) heuristic is based on movements within the local neighborhood with respect to location and allocation decisions and routing has to be calculated optimally with respect to these changes. However, as discussed in Chapters 1 and 4, the resulting routing problem, the WDMP, turns out to be quite difficult to solve as no known polynomial time algorithm exists to solve it on general networks. Therefore, we did not provide detailed discussions for the LAR heuristic. The two other heuristic approaches, namely the LRA and the ARL, are studied in detail and two different algorithms are developed for each heuristic. The first algorithm updates the current feasible solution as soon as another solution which improves the objective function is found, whereas the second algorithm searches for the best improvement within the given local neighborhood before updating the current solution. The two versions of each heuristic are tested on randomly generated networks and their performances are compared with respect to computation time and closeness to optimality whenever optimal solutions were available. For larger problem instances where the branch-and-bound algorithm was not able to solve the problem within reasonable time (we terminated the algorithm when computation time exceeded 1 hour of CPU time on Sun Sparc V with 70mHz processor), the heuristics are compared with each other with respect to computation time, closeness to best solution found, and the number of times the heuristic was able to find the best solution.

As a result of computational testing and performance comparison of heuristics, LRA type heuristics seem to outperform ARL type heuristics both in computation time and closeness to optimality (or best solution found). Although the second version of LRA heuristic, LRA(B), which searches for best improvement

within the given neighborhood before updating the current solution, does not seem to perform better than the first version, LRA(A), for smaller problem instances, as the problem size increases, it begins to outperform LRA(A) with respect to the number of times each algorithm found the best solution. Moreover, it does not take significantly longer computation time than LRA(A).

In the light of the research performed in this study, we will discuss directions for future research in the next section.

5.2 Directions for Future Research

The problem considered in this study is introduced as a new model for a location and distribution system. Thus, the research presented here is a first attempt to solve the problem. There are many aspects of the problem still open to question. Next we will discuss several directions for future research in connection with some of those questions.

5.2.1 The Weighted Delivery Man Problem (WDMP)

As it was discussed in Chapter 1, when the locations of facilities and the allocations of demand points to facilities are known, the remaining problem related to the routing decision is the Weighted Delivery Man Problem. The problem is discussed briefly with its relation to the well-known Delivery Man Problem. It appears that this problem has not been previously studied in the literature either. Therefore the WDMP might be a good direction for future research. Although the problem can be reduced to the DMP when weights associated with the demand points are integers or rational numbers, the problem size may increase very rapidly if the relative size of weights

is large. Therefore, we believe that studying the problem in its original formulation may provide better solution procedures and also insights into the inherent difficulties of the problem. The problem itself is quite difficult as no known polynomial time algorithm exists to solve its special case, the DMP , when all weights are equal, even on tree networks.

Additionally, better and efficient solution procedures for the WDMP may lead to better heuristics for SCLAP as well. For example, the LAR heuristic discussed in Chapter 4 can utilize algorithms developed to solve the WDMP at each iteration of the local search procedure.

5.2.2 Variations of SCLAP

Clearly, one can come up with many variations of SCLAP which can be studied in the future. For example, the facilities may have capacities imposed on them, or multiple commodities with different amounts of demand at each customer point has to be distributed along the same route. Additionally, shipments from facilities to demand points may be done along subroutes, instead of direct shipments. There may be more than one vehicle to serve facilities and the demand points, which requires determining more than one supply route, etc. Further variations can be studied for cases where the demands are stochastic or the presence of a competitor in the market affect the decisions of the company. Moreover, combinations of those variations can bring up even more complex versions of the problem.

One immediate variation of SCLAP rises when the number of facilities to locate is known beforehand and instead of including facility location costs in the model implicitly through constraining the number of facilities to locate, there may be

different fixed cost values for facility location at each candidate site. This variation is discussed in Chapter 1 briefly. As it was pointed out in Chapter 1, this problem reduces to the uncapacitated facility location problem when the cost of shipment on the supply route is ignored. This classical problem is discussed in Section 1.2.2 of Chapter 1. Because of its obvious similarities to the p-median problem, in general, the solution procedures developed for both problems in the literature are very similar in nature. With this observation in mind, we believe that the solution approaches to SCLAP may lead to relatively straightforward extensions to this variation and vice versa.

5.2.3 Effect of Scaling Factor α

In the model we studied, we assume that the cost of shipment on the supply route will be cheaper, because in most business applications, shipments between facilities would be done in bulk and therefore be less costly. This is represented by multiplying the cost of shipment on the supply route by a scaling factor α, $0 < \alpha < 1$, and it is assumed to be uniform on all arcs of the network. The validity of this assumption may be challenged when the reduction in cost varies along different parts of the route.

Moreover, this factor has a significant effect on the problem in its current form as well. First, it changes the allocation sets for each facility as it varies between 0 and 1. For example, when α is closer to 0, one would expect that the demand points would be allocated to facilities more uniformly, resembling a solution the the pure p-median problem. In those cases, the solution to the associated p-median problem may provide a good approximation of the solution to the original problem or it may provide good initial starting points for local search type heuristics developed for SCLAP. On

the other hand, when α is closer to 1, one might expect that more demand points will be allocated to facilities that are visited earlier on the supply route. In those cases, construction type heuristics that start with no facilities open and add new facilities one by one until p facilities are open, may give better approximations of the optimal solution to SCLAP or provide good starting points for local search heuristics for SCLAP.

5.2.4 Tighter Bounds on the Optimal Objective Function Value

A simple and immediate lower bound on the optimal objective function value of SCLAP is obtained by assuming that all demand points will receive their shipment from the supply plant directly with the discounted rate, as discussed in Chapter 3. For better comparison of heuristics for large problem instances where an optimal solution is not available, tighter and efficiently solvable bounds have to be developed to provide tighter estimates of the difference of the heuristic solutions from a potential optimal solution. Solutions to the subproblems of SCLAP or bounds already developed for those problems may have potential uses for development of bounds for the SCLAP itself.

5.2.5 Mathematical Programming Approaches

In this study, we apply a combinatorial approach to solving the SCLAP. Mathematical Programming formulations of SCLAP, like many other location-routing problems, may be quite complex, in that they involve too many variables and constraints as well as nonlinearity in the objective function and/or in the constraints. Despite

this disadvantage of the Mathematical Programming approach, it may still be worth investigating for good solution approaches to the SCLAP, through utilization of relaxations and Lagrangian approaches. Insights gained from those formulations may lead to development of relaxation approaches and consequently, development of a branch-and-bound type algorithm based on bounds obtained through mathematical programming relaxations.

5.2.6 Improvements on the Local Search Heuristics

The main disadvantage of local search type heuristics, discussed in Chapter 4, is that they find solutions which are locally optimum. Thus, when the algorithm falls into an unfavorable neighborhood, it may find very poor solutions which might be quite far from the global optimum. Two approaches are used traditionally, to avoid "bad" local optima. One is to start the algorithm from many starting points uniformly distributed over the feasible solution space. Probabilistically, the more initial starting points are used the more it is likely that a better solution will be found. This was addressed in our studies by initializing the problem over 10 different starting points and selecting the best solution found among all of them. Another approach used in the literature, is to allow random "jumps" to a worse solution to avoid the neighborhood of bad local optima. Simulated Annealing, Tabu Search, Genetic Algorithms, Neural Networks are different approaches that are used to allow such random "jumps" through a controlled process. Those approaches can also be applied to the local search heuristics developed for SCLAP.

5.3 Conclusion

We introduced a new model in this study and attempted a step forward towards solving the problem. There are many aspects of the problem still open to question, some of which are discussed in the preceding section. The next step for us is to try to further improve on the work presented here.

Appendix A

The Weighted Delivery Man Problem on Trees

In this appendix, we present some new results for the Weighted Delivery Man Problem (WDMP) on tree networks.

The problem was formulated in Chapter 1, as follows:

$$Min \quad \sum_{j=0}^{n-1} \left(\sum_{i=j+1}^{n} w_i \right) d(v_j, v_{j+1}). \tag{A.1}$$

where the function $d(\cdot, \cdot)$ represents the distance between two points and w_i denote the weight (or amount of demand) at node v_i.

Now, let T be a tree rooted at node v_0 with node set V and edge set E. We wish to find a tour starting at v_0 which minimizes the total weighted waiting time in the system. Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan, and Sudan (1994) presented an algorithm which gave an approximation ratio of 3.5912 for the DMP on tree networks. The algorithm is based on combining TSP routes of 1,2,...,n nodes on the tree in an intelligent way to get the final route for DMP. TSP is trivial on tree networks, where any depth-first traversal of the edges give an optimal solution. In this section, we will present a lower bound for the WDMP using k-shortest paths,

instead of TSP routes on trees. The k- shortest path problem (k-SPP) involves finding the shortest path between an origin and a destination, visiting k-nodes. This problem is NP-Hard on general networks. However,it is can be solved in polynomial time on tree networks..

Assume that we solve k-SPP on V for $k = 1, 2, .., n-1$. All paths start at v_0 and end in one of the nodes of the network. Denote the paths so obtained by $P_1(v_j)$, $P_2(v_j)$, ... , $P_{n-1}(v_j)$ $\forall v_j \in V$. Let $D_k(v_j)$, $\forall v_j \in V$ and $k = 1, 2, .., n-1$, denote the length of the path $P_k(v_j)$.

Now, the lower bound on the optimal objective function value of the WDMP is given by

Theorem A.1 *Let z^* denote the optimum objective value of WDMP on T. Then*

$$\sum_{i=1}^{n-1} \min_{v_j \in V} w_j D_i(v_j) + 2w_n \sum_{(v_i, v_j) \in E} d(v_i, v_j) \leq z^*. \tag{A.2}$$

Proof: Let P^* be the optimal tour for WDMP. Let P_k^*, $k = 1, ..., n$ denote the portions of the optimal tour that start from the origin and end at the k-th node on the tour. Let v_k^* denote the last node on the path P_k^*. Let $D_k^*(v_k^*)$ denote the distance traveled on the optimum tour up to the k-th node. Then the optimal objective value z^* is given by

$$z^* = \sum_{k=1}^{n} w_k^* D_k^*(v_k^*). \tag{A.3}$$

First, we have

$$D_k(v_k^*) \leq D_k^*(v_k^*). \tag{A.4}$$

This is because $D_k(v_k^*)$ is defined to be the length of the k-shortest path from origin v_0 to the vertex v_k^* visiting k nodes. Thus any other path between v_0 and v_k^* passing

through k nodes has to be longer in length. This argument is true for all the nodes in the optimal tour.

For the last node to be visited on the optimal tour (that is the origin), we take the solution to the TSP on the given tree. Since the optimum TSP tour on a tree is any depth-first tour, in which the nodes are visited according to the depth-first rule and each edge is traversed twice, the length of the tour is two times the length of the tree (i.e. sum of all edge lengths). So, we have

$$2 \sum_{(v_i,v_j)\in E} d(v_i, v_j) \leq D_n^*(v_n). \tag{A.5}$$

From (A.3) and because the weights are nonnegative, we know that

$$w_k^* D_k(v_k^*) \leq w_k^* D_k^*(v_k^*). \tag{A.6}$$

It is then straightforward to show that

$$\min_{v_j\in V} w_j D_j(v_j) \leq w_k^* D_k(v_k^*) \leq w_k^* D_k^*(v_k^*). \tag{A.7}$$

Clearly, the lower bound given in the theorem now follows directly from (A.5) and (A.6). That is

$$\sum_{k=1}^{n-1} \min_{v_j\in V} w_j D_i(v_j) + 2w_n \sum_{(v_i,v_j)\in E} d(v_i, v_j) \leq \sum_{k=1}^{n} w_k^* D_k^*(v_k^*) = z^* \tag{A.8}$$

Thus, the proof is complete. □

Note that, when the first k vertices $\{v_1^*, ..v_k^*\}$ to be visited on the optimal tour are already known, then the lower bound can be updated as follows:

Corollary A.1 *Let the first k vertices $\{v_1^*, ..v_k^*\}$ to be visited on the optimal tour be fixed beforehand, then the lower bound on the optimal objective function value is given*

by

$$\sum_{i=1}^{k} w_k^* D_k^*(v_k^*) + \sum_{i=k+1}^{n-1} \min_{v_j \in V-\{v_1^*,\ldots,v_k^*\}} w_j \left(D_k^*(v_k^*) + D_{i-k}^k(v_j)\right) + 2w_n \sum_{(v_i,v_j)\in E} d(v_i, v_j) \leq z^*. \tag{A.9}$$

where $D_{i-k}^k(v_j)$ *denotes the* $(i-k)$*-shortest path from the origin* v_k^* *to a vertex* v_j.

The above result follows from the observation that when first k nodes on the optimal tour are known, then we can calculate the $(i-k)$-shortest paths from the k-th vertex of the tour to the rest of the vertices that are not yet visited and still have

$$D_k^*(v_k^*) + D_{i-k}^k(v_k^*) \leq D_i^*(v_i^*) \tag{A.10}$$

The rest of the analysis follows a similar fashion given in the proof of Theorem A.1.

Once we have the given updating mechanism for a lower bound, we can devise a branch and bound algorithm that will solve the problem. We start with the root node (or the origin) v_0. At each step we visit a candidate vertex and update the lower bound. If at any stage the lower bound becomes greater than the upper bound that branch is pruned. For a starting upper bound we can choose an arbitrary sequence of vertices and calculate the objective function value. Every time we reach a whole sequence of vertices in the B&B tree, we calculate the objective function value. If the objective value of the new sequence is less than the current upper bound, we update the upper bound. We continue until all branches of the B&B tree are either pruned or traversed.

This approach can also be extended to general networks. However, the k-shortest path problem is not polynomially solvable on general networks. Thus the branch and bound algorithm may quickly lead to inefficient computation times.

Appendix B

Comparisons for Each Test Problem: Exact vs Heuristic

In Chapter 4, the comparisons of heuristics with respect to closeness to optimality or the best solution found and the computation times were discussed via average performances over 10 problems for each n, p, α combination. In this and the following two appendices, we provide the comparisons for each individual test problem.

For each value of n (10, 20, 30, 40, 50), 10 different networks were randomly generated consisting of n demand points plus one supply plant node. The edge lengths and weights at the nodes were drawn from a uniform distribution between 1 and 100. Each test problem for each n was then solved for different p and α values.

In this appendix, we compare the performances of heuristics on each of the test problems with the branch and bound algorithm which gives the optimum solution. Section B.1 reports the results for percentage error, that is, how far the solution found by the heuristic is away from the actual optimum. Section B.2 provides the respective computation times for each heuristic and the B&B algorithm.

B.1 Comparisons for Closeness to Optimality

The tables in the following pages clearly indicate the dominance of heuristics of type LRA over the type ARL. In most cases both algorithms of LRA type heuristics found solutions within about 2% of the actual optimum. However, of course there were a few instances where the performance was much poorer. For example, as depicted in Table B.1, Algorithm LRA(B) found a solution which was 10.95% far from the actual optimum for $p = 1$ and $\alpha = 0.1$ for the first test problem with $n = 10$ demand points. ARL heuristics performed a lot poorer compared to LRA type heuristics, they found solutions that were as much as about 19% away from the optimum (see Table B.33). For the problem sizes in this appendix, LRA(A) seemed to perform better than LRA(B). Although ARL(B) seemed to perform better than ARL(A), there was a discrepancy in their performance against each other quite often. Even for the same test problem, one would perform better than the other for certain p and α values, and yet perform poorly for other values of p and α.

B.1.1 Test Problems with 10 Demand Points

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	8084.5	0.00	10.95	15.39	0.00
10	3	0.2	9786.0	0.00	0.00	12.41	13.13
10	3	0.3	11487.5	0.00	0.93	4.41	10.47
10	3	0.4	13189.0	0.00	0.00	9.42	9.08
10	3	0.5	14814.5	0.00	1.35	6.73	6.73
10	3	0.6	16394.4	0.00	0.00	4.87	4.87
10	3	0.7	17974.3	0.00	0.81	4.25	1.00
10	3	0.8	19554.2	0.00	0.00	3.42	3.44
10	3	0.9	20994.1	0.00	0.00	0.57	0.57
10	5	0.1	5965.5	0.00	0.00	0.00	3.75
10	5	0.2	8472.4	0.00	0.00	0.00	4.22
10	5	0.3	10604.6	0.00	0.00	6.17	0.00
10	5	0.4	12664.0	0.00	0.00	5.27	3.04
10	5	0.5	14377.0	0.00	0.00	6.26	3.55
10	5	0.6	16044.4	0.00	0.00	7.78	4.25
10	5	0.7	17711.8	0.00	0.00	8.22	4.82
10	5	0.8	19379.2	0.00	0.00	6.27	4.44
10	5	0.9	20956.6	0.00	0.00	7.30	3.99

Table B.1: % Error: Test Problem No. 1 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	7275.5	0.00	0.00	3.65	0.00
10	3	0.2	9481.0	0.00	0.00	4.18	0.00
10	3	0.3	11659.0	0.00	0.00	4.98	3.06
10	3	0.4	13837.0	0.00	0.00	3.90	0.91
10	3	0.5	15562.0	0.00	0.00	2.37	2.37
10	3	0.6	17161.0	0.00	0.00	2.65	1.94
10	3	0.7	18760.0	0.00	0.00	1.94	1.63
10	3	0.8	20359.0	0.00	0.00	1.65	0.00
10	3	0.9	21958.0	0.00	0.00	1.24	0.00
10	5	0.1	4793.8	1.13	0.00	1.13	0.00
10	5	0.2	8314.6	0.00	0.00	0.22	0.22
10	5	0.3	11131.0	4.74	0.22	0.00	0.39
10	5	0.4	13405.0	3.22	0.00	0.66	0.31
10	5	0.5	15464.5	0.00	0.00	0.35	3.25
10	5	0.6	17083.0	0.00	0.00	0.62	0.44
10	5	0.7	18701.5	0.00	0.00	0.84	0.72
10	5	0.8	20320.0	0.00	0.00	1.05	1.08
10	5	0.9	21938.5	0.00	0.00	1.16	1.16

Table B.2: % Error: Test Problem No. 2 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	4479.1	0.00	0.00	2.01	2.01
10	3	0.2	6388.2	0.00	0.00	0.00	0.00
10	3	0.3	8297.3	0.00	0.00	5.45	1.35
10	3	0.4	10206.4	0.00	0.00	8.95	0.00
10	3	0.5	12115.5	0.00	0.00	7.21	0.00
10	3	0.6	14024.6	0.00	2.30	3.28	3.28
10	3	0.7	15933.7	0.00	0.00	3.29	0.94
10	3	0.8	17842.8	0.00	0.59	0.90	0.90
10	3	0.9	19653.9	0.00	0.00	0.27	0.14
10	5	0.1	3270.6	0.00	0.00	0.00	0.00
10	5	0.2	5619.8	0.00	0.00	0.00	0.00
10	5	0.3	7776.2	0.00	0.00	1.53	0.00
10	5	0.4	9932.6	0.00	0.00	0.89	0.00
10	5	0.5	11956.5	0.04	0.04	4.66	1.11
10	5	0.6	13897.4	0.03	0.03	0.00	1.13
10	5	0.7	15838.3	0.02	0.02	2.11	0.00
10	5	0.8	17779.2	0.01	0.00	1.25	1.47
10	5	0.9	19622.1	0.01	0.01	1.21	0.68

Table B.3: % Error: Test Problem No. 3 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	6901.4	0.00	0.00	3.88	0.00
10	3	0.2	8174.8	0.00	0.00	2.50	0.00
10	3	0.3	9448.2	0.00	1.26	7.34	7.34
10	3	0.4	10713.4	0.00	0.00	6.18	0.00
10	3	0.5	11859.5	0.00	0.00	3.46	3.46
10	3	0.6	12944.8	0.00	0.00	8.15	5.36
10	3	0.7	13878.1	0.00	0.00	14.78	8.34
10	3	0.8	14811.4	0.00	0.00	3.28	0.00
10	3	0.9	15744.7	0.00	0.00	0.63	5.97
10	5	0.1	4734.2	0.00	0.00	0.16	0.00
10	5	0.2	6462.4	0.00	0.00	13.30	0.00
10	5	0.3	8190.6	0.00	0.00	0.00	0.00
10	5	0.4	9918.8	0.00	0.00	0.00	0.12
10	5	0.5	11248.5	0.00	0.00	3.54	3.54
10	5	0.6	12547.2	0.00	0.00	4.39	4.97
10	5	0.7	13785.1	0.00	0.00	3.30	0.44
10	5	0.8	14749.4	0.00	0.00	3.79	5.90
10	5	0.9	15713.7	0.00	0.00	4.22	5.98

Table B.4: % Error: Test Problem No. 4 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	3296.2	0.00	0.00	0.55	0.00
10	3	0.2	4638.6	0.00	0.00	5.83	5.83
10	3	0.3	5858.5	1.65	0.00	9.73	4.59
10	3	0.4	6902.0	0.00	0.00	6.87	0.27
10	3	0.5	7945.5	0.00	1.41	5.23	1.97
10	3	0.6	8989.0	0.00	2.28	4.56	1.25
10	3	0.7	10032.5	0.00	1.53	3.06	0.84
10	3	0.8	11076.0	0.00	0.92	1.94	0.92
10	3	0.9	12119.5	0.00	0.00	0.55	0.90
10	5	0.1	2264.0	0.00	0.00	0.00	0.00
10	5	0.2	3843.0	0.00	0.00	4.02	0.00
10	5	0.3	5304.9	0.00	0.00	0.44	0.00
10	5	0.4	6633.2	0.00	0.00	0.41	2.39
10	5	0.5	7730.0	0.00	0.00	0.39	0.21
10	5	0.6	8816.6	0.00	0.00	0.37	0.65
10	5	0.7	9903.2	0.00	0.00	2.97	2.97
10	5	0.8	10989.8	0.00	0.00	2.12	0.00
10	5	0.9	12076.4	0.00	0.00	1.32	0.92

Table B.5: % Error: Test Problem No. 5 (n=10)

n	p	α	Optimal Obj. Val.	% Error LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	6593.1	0.00	0.00	0.00	0.00
10	3	0.2	8592.2	0.00	0.00	0.00	0.00
10	3	0.3	9770.7	0.00	0.00	14.90	14.31
10	3	0.4	10866.6	0.00	9.01	7.58	6.07
10	3	0.5	11962.5	0.00	0.00	9.20	6.95
10	3	0.6	13058.4	0.00	2.66	9.12	6.74
10	3	0.7	14154.3	0.00	0.00	3.10	4.84
10	3	0.8	15250.2	0.00	0.00	3.50	2.47
10	3	0.9	16346.1	0.00	0.00	2.10	1.35
10	5	0.1	4996.7	0.00	0.00	0.00	0.00
10	5	0.2	7644.8	0.00	0.00	1.36	0.00
10	5	0.3	9469.8	0.00	0.00	1.27	0.00
10	5	0.4	10672.2	0.00	0.00	3.62	4.54
10	5	0.5	11800.5	0.00	0.00	3.80	3.80
10	5	0.6	12928.8	0.00	0.00	6.91	5.49
10	5	0.7	14057.1	0.00	0.00	7.28	6.91
10	5	0.8	15185.4	0.00	0.00	7.02	7.02
10	5	0.9	16313.7	0.00	0.00	6.06	7.14

Table B.6: % Error: Test Problem No. 6 (n=10)

n	p	α	Optimal Obj. Val.	% Error LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	3608.2	0.00	0.00	0.41	0.00
10	3	0.2	5057.2	0.00	0.00	0.89	0.38
10	3	0.3	5781.3	0.00	0.00	0.49	13.60
10	3	0.4	6505.4	0.00	0.00	5.69	3.37
10	3	0.5	7229.5	0.00	0.00	10.68	10.81
10	3	0.6	7765.0	0.00	0.00	2.88	1.93
10	3	0.7	8260.0	0.00	0.00	6.95	0.00
10	3	0.8	8755.0	0.00	0.00	4.21	3.76
10	3	0.9	9250.0	0.00	0.00	1.53	1.53
10	5	0.1	2672.9	0.00	0.00	0.00	0.00
10	5	0.2	4194.8	0.00	0.00	0.00	0.00
10	5	0.3	5516.7	0.00	0.00	0.75	0.75
10	5	0.4	6278.6	0.00	0.00	1.23	0.75
10	5	0.5	7040.5	0.00	0.00	5.70	0.88
10	5	0.6	7765.0	0.00	0.00	4.53	1.48
10	5	0.7	8260.0	0.00	0.00	6.88	5.38
10	5	0.8	8755.0	0.00	0.00	4.78	4.97
10	5	0.9	9250.0	0.00	0.00	2.93	2.78

Table B.7: % Error: Test Problem No. 7 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	5040.5	0.00	0.64	0.00	0.00
10	3	0.2	6290.0	0.00	0.64	0.00	0.00
10	3	0.3	7434.0	0.00	0.00	0.00	0.00
10	3	0.4	8285.0	0.00	0.00	0.00	0.00
10	3	0.5	9089.5	0.00	0.00	7.72	1.43
10	3	0.6	9894.0	0.00	1.13	15.46	6.32
10	3	0.7	10698.5	0.00	0.00	3.79	0.00
10	3	0.8	11503.0	0.00	0.00	0.45	0.45
10	3	0.9	12307.5	0.00	0.00	1.71	0.00
10	5	0.1	3506.2	0.00	0.00	0.00	0.00
10	5	0.2	5349.2	0.00	0.00	3.89	0.00
10	5	0.3	6838.3	0.00	0.00	0.00	0.44
10	5	0.4	8027.4	0.00	0.00	2.79	2.79
10	5	0.5	9011.5	0.00	0.00	2.27	2.44
10	5	0.6	9831.6	0.00	0.00	3.54	3.98
10	5	0.7	10651.7	0.00	0.00	4.46	4.46
10	5	0.8	11471.8	0.00	0.00	5.25	5.25
10	5	0.9	12291.9	0.00	0.00	5.94	6.55

Table B.8: % Error: Test Problem No. 8 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	11632.1	0.00	0.00	0.00	0.00
10	3	0.2	15451.4	0.00	0.00	0.00	0.38
10	3	0.3	19209.6	0.00	0.00	0.50	0.54
10	3	0.4	22967.8	0.00	0.00	0.00	0.22
10	3	0.5	26548.0	0.00	0.00	0.18	0.13
10	3	0.6	29946.6	0.00	0.00	0.98	0.98
10	3	0.7	33345.2	0.00	0.00	0.00	1.43
10	3	0.8	36743.8	0.00	0.00	0.66	0.68
10	3	0.9	40142.4	0.00	0.00	1.19	0.40
10	5	0.1	8390.0	0.00	0.00	0.00	0.00
10	5	0.2	13413.6	0.00	0.00	1.39	0.00
10	5	0.3	17440.4	0.00	0.00	1.88	0.00
10	5	0.4	21462.2	0.00	0.00	0.00	1.14
10	5	0.5	25484.0	0.00	0.86	1.20	0.46
10	5	0.6	29271.0	0.00	0.00	0.80	1.99
10	5	0.7	32838.5	0.00	0.00	0.40	1.36
10	5	0.8	36406.0	0.00	0.00	1.19	0.72
10	5	0.9	39973.5	0.00	0.00	0.00	0.00

Table B.9: % Error: Test Problem No. 9 (n=10)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	5186.6	0.00	0.86	0.00	0.00
10	3	0.2	7059.2	0.00	0.00	0.00	0.00
10	3	0.3	8804.6	0.00	0.00	0.00	0.00
10	3	0.4	10421.8	0.00	0.00	0.65	0.00
10	3	0.5	12039.0	0.00	0.00	0.00	0.42
10	3	0.6	13656.2	0.00	0.00	3.04	0.00
10	3	0.7	15092.7	0.00	0.00	0.64	0.00
10	3	0.8	16436.8	0.00	0.00	2.38	1.85
10	3	0.9	17780.9	0.00	0.00	1.09	0.97
10	5	0.1	3593.9	0.00	0.00	0.00	0.00
10	5	0.2	5846.2	0.00	0.00	0.00	0.00
10	5	0.3	7959.7	0.00	1.29	1.96	0.00
10	5	0.4	9831.4	0.00	0.00	1.68	0.33
10	5	0.5	11547.0	0.00	0.00	3.46	0.23
10	5	0.6	13261.4	0.00	0.01	0.25	0.00
10	5	0.7	14873.4	0.04	0.04	1.37	0.00
10	5	0.8	16290.6	0.03	0.03	0.52	0.03
10	5	0.9	17707.8	0.01	0.01	0.54	0.00

Table B.10: % Error: Test Problem No. 10 (n=10)

B.1.2 Test Problems with 20 Demand Points

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	6884.6	0.00	0.00	8.03	8.03
20	3	0.2	9380.6	0.00	4.47	6.48	5.39
20	3	0.3	11830.5	0.00	2.73	4.70	5.85
20	3	0.4	14252.0	0.00	0.01	5.43	5.54
20	3	0.5	16629.0	0.00	0.00	4.22	3.70
20	3	0.6	18966.8	0.00	0.00	3.48	3.48
20	3	0.7	21199.1	0.00	0.00	2.49	3.13
20	3	0.8	23344.4	0.00	0.00	1.82	0.72
20	3	0.9	25445.5	0.00	0.13	0.81	0.75
20	5	0.1	5766.2	0.00	0.00	1.86	3.32
20	5	0.2	8587.6	0.06	0.06	1.36	1.76
20	5	0.3	11196.7	0.07	0.07	2.31	4.33
20	5	0.4	13677.6	0.00	0.27	2.69	1.27
20	5	0.5	16118.5	0.00	0.00	2.86	3.64
20	5	0.6	18475.6	0.00	0.00	2.84	3.27
20	5	0.7	20827.7	0.13	0.23	2.68	1.96
20	5	0.8	23141.2	0.25	0.29	1.90	1.57
20	5	0.9	25326.6	0.00	0.00	1.05	0.92

Table B.11: % Error: Test Problem No. 1 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	9391.4	0.00	0.00	13.25	8.32
20	3	0.2	11631.4	0.00	0.00	8.43	9.52
20	3	0.3	13422.8	0.00	1.33	9.68	6.00
20	3	0.4	15146.6	0.00	0.00	9.84	7.26
20	3	0.5	16853.0	0.00	0.00	7.80	6.08
20	3	0.6	18528.2	0.00	0.86	5.36	5.36
20	3	0.7	20101.7	0.00	0.00	3.57	2.75
20	3	0.8	21629.4	0.00	0.06	2.97	2.82
20	3	0.9	23037.7	0.00	0.03	2.94	2.86
20	5	0.1	7740.0	0.00	0.00	0.00	0.00
20	5	0.2	10170.4	0.00	0.00	3.60	3.80
20	5	0.3	12439.1	0.00	0.00	3.15	0.53
20	5	0.4	14469.0	0.00	0.04	4.19	3.25
20	5	0.5	16322.0	0.00	0.00	1.15	1.15
20	5	0.6	18072.2	0.00	0.00	8.70	3.33
20	5	0.7	19775.0	0.00	0.00	4.07	3.37
20	5	0.8	21431.0	0.00	0.00	4.28	1.93
20	5	0.9	22966.9	0.00	0.00	1.55	0.20

Table B.12: % Error: Test Problem No. 2 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	10444.4	0.00	0.00	0.00	3.25
20	3	0.2	11688.8	0.00	0.00	12.15	3.09
20	3	0.3	12899.9	0.00	0.00	4.70	4.44
20	3	0.4	14059.2	0.00	0.00	10.83	7.59
20	3	0.5	15174.5	0.00	0.00	9.89	9.89
20	3	0.6	16131.4	0.00	0.00	13.75	13.47
20	3	0.7	17079.5	0.00	0.05	15.25	6.52
20	3	0.8	17766.0	0.00	1.20	10.82	5.91
20	3	0.9	18452.5	0.00	0.58	1.33	2.00
20	5	0.1	7888.4	5.06	0.00	0.00	6.38
20	5	0.2	10068.8	0.00	0.00	6.53	5.36
20	5	0.3	11807.2	0.00	1.68	7.08	0.00
20	5	0.4	13122.6	0.00	2.82	7.50	13.04
20	5	0.5	14438.0	0.00	0.00	0.00	11.63
20	5	0.6	15656.6	0.00	0.62	6.65	5.21
20	5	0.7	16732.2	0.00	0.00	6.10	4.88
20	5	0.8	17666.0	0.00	0.00	3.50	3.50
20	5	0.9	18402.5	0.00	0.00	2.35	2.24

Table B.13: % Error: Test Problem No. 3 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	9077.5	0.00	0.00	0.49	0.96
20	3	0.2	11144.4	0.00	1.95	7.04	6.71
20	3	0.3	13099.5	0.52	0.52	7.73	7.10
20	3	0.4	14911.0	0.00	1.50	7.80	8.08
20	3	0.5	16704.5	0.00	1.13	6.33	5.63
20	3	0.6	18462.0	0.00	0.81	6.41	5.11
20	3	0.7	20178.0	0.00	0.46	4.88	3.60
20	3	0.8	21811.0	0.00	0.77	3.52	2.86
20	3	0.9	23089.1	0.09	0.18	1.36	1.24
20	5	0.1	6580.3	0.00	0.00	2.55	5.01
20	5	0.2	9633.8	0.00	0.00	1.59	3.95
20	5	0.3	11857.8	0.00	0.00	1.88	2.24
20	5	0.4	13848.4	0.00	0.00	3.24	2.15
20	5	0.5	15821.0	0.00	0.00	4.76	7.23
20	5	0.6	17755.2	0.00	0.00	2.06	2.18
20	5	0.7	19689.4	0.00	0.00	2.08	2.03
20	5	0.8	21594.2	0.14	0.14	2.04	2.13
20	5	0.9	23063.5	0.11	0.03	0.21	0.03

Table B.14: % Error: Test Problem No. 4 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	9866.6	0.00	0.00	1.77	4.36
20	3	0.2	10600.4	0.00	0.00	9.95	0.11
20	3	0.3	11325.6	0.00	0.00	11.07	6.86
20	3	0.4	11876.8	0.00	0.21	6.09	9.46
20	3	0.5	12337.0	0.00	1.42	8.19	7.01
20	3	0.6	12797.2	0.00	1.09	9.33	6.66
20	3	0.7	13257.4	0.00	0.00	11.12	11.90
20	3	0.8	13717.6	0.00	0.39	6.15	6.15
20	3	0.9	14177.8	0.00	0.25	3.20	5.44
20	5	0.1	7733.9	0.00	0.00	3.22	3.22
20	5	0.2	9314.0	0.00	0.00	5.04	3.38
20	5	0.3	10310.9	0.12	0.00	9.73	7.04
20	5	0.4	11179.2	0.48	1.03	5.15	5.82
20	5	0.5	11863.0	0.00	0.00	3.76	4.03
20	5	0.6	12431.6	0.00	0.00	3.50	3.38
20	5	0.7	13000.2	0.00	0.00	3.30	1.48
20	5	0.8	13568.8	0.00	0.00	1.52	0.64
20	5	0.9	14123.2	0.06	0.06	0.85	0.40

Table B.15: % Error: Test Problem No. 5 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	7733.0	0.00	0.00	0.00	0.00
20	3	0.2	9538.2	0.00	0.00	4.71	2.60
20	3	0.3	11148.0	0.00	1.43	3.15	2.19
20	3	0.4	12220.6	0.00	0.00	6.37	5.59
20	3	0.5	13172.5	0.00	0.00	5.39	8.64
20	3	0.6	14111.2	0.00	0.00	4.67	4.10
20	3	0.7	15049.9	0.00	1.54	7.67	7.67
20	3	0.8	15988.6	0.00	0.00	4.42	4.42
20	3	0.9	16927.3	0.00	0.46	2.25	1.35
20	5	0.1	6189.5	0.00	0.78	1.42	0.00
20	5	0.2	8577.6	0.00	0.00	0.93	0.09
20	5	0.3	10276.6	0.00	0.93	5.95	6.59
20	5	0.4	11698.4	1.20	0.43	6.01	5.35
20	5	0.5	12846.5	0.35	0.35	0.35	4.00
20	5	0.6	13900.8	0.00	0.26	4.49	4.34
20	5	0.7	14915.5	0.51	0.45	5.85	4.86
20	5	0.8	15899.0	0.32	0.32	4.61	2.64
20	5	0.9	16882.5	0.15	0.02	1.24	2.54

Table B.16: % Error: Test Problem No. 6 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	10939.6	0.00	0.52	2.95	1.45
20	3	0.2	11696.8	0.00	0.45	5.41	1.98
20	3	0.3	12377.7	0.00	1.58	6.41	5.07
20	3	0.4	12984.4	0.00	0.16	8.07	6.52
20	3	0.5	13522.5	0.00	2.39	3.39	4.17
20	3	0.6	14044.0	0.00	1.86	4.39	3.17
20	3	0.7	14524.0	0.00	1.37	5.93	4.94
20	3	0.8	15004.0	0.00	0.00	3.39	3.39
20	3	0.9	15484.0	0.00	0.43	1.30	1.51
20	5	0.1	8824.4	0.00	0.00	2.49	4.06
20	5	0.2	10459.4	0.40	0.00	3.79	4.48
20	5	0.3	11479.5	1.84	2.03	6.20	3.83
20	5	0.4	12400.0	2.42	1.17	1.17	2.98
20	5	0.5	13260.5	0.38	0.13	1.29	3.20
20	5	0.6	13891.8	0.00	0.00	2.53	2.44
20	5	0.7	14422.3	0.00	0.00	1.24	1.40
20	5	0.8	14936.2	0.00	0.00	3.04	1.57
20	5	0.9	15450.1	0.00	0.00	2.31	0.45

Table B.17: % Error: Test Problem No. 7 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	8608.0	0.00	0.00	0.00	0.00
20	3	0.2	10402.0	0.00	0.00	0.16	6.59
20	3	0.3	12170.6	0.00	0.00	6.07	0.70
20	3	0.4	13613.8	0.00	0.00	12.86	8.77
20	3	0.5	15057.0	0.00	0.00	3.13	3.35
20	3	0.6	16437.2	0.00	0.00	4.84	5.55
20	3	0.7	17685.4	0.00	0.00	3.24	1.45
20	3	0.8	18864.6	0.00	0.00	3.25	3.25
20	3	0.9	20043.8	0.00	0.00	1.37	0.71
20	5	0.1	7172.5	0.14	0.23	1.08	3.36
20	5	0.2	9178.0	0.00	0.16	2.44	5.13
20	5	0.3	11115.1	0.00	0.53	7.48	4.05
20	5	0.4	12927.4	1.80	0.00	7.84	5.52
20	5	0.5	14485.0	1.34	0.00	8.98	3.93
20	5	0.6	16001.2	0.00	0.16	6.44	1.64
20	5	0.7	17420.8	0.16	0.49	1.50	3.61
20	5	0.8	18688.2	0.30	0.30	3.56	1.93
20	5	0.9	19955.6	0.14	0.14	0.78	1.72

Table B.18: % Error: Test Problem No. 8 (n=20)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	12040.5	0.00	0.00	10.16	3.00
20	3	0.2	14092.6	0.00	0.00	10.34	11.19
20	3	0.3	15961.1	0.00	0.00	3.66	7.17
20	3	0.4	17568.6	0.00	0.00	6.68	6.88
20	3	0.5	19171.5	0.00	0.50	7.02	7.02
20	3	0.6	20774.4	0.00	0.70	2.87	3.44
20	3	0.7	22338.5	0.00	0.06	2.89	2.89
20	3	0.8	23732.8	0.00	0.00	0.72	3.55
20	3	0.9	25084.4	0.00	0.00	2.13	1.58
20	5	0.1	9148.2	0.00	0.00	3.75	5.37
20	5	0.2	12117.8	0.00	0.00	13.17	5.48
20	5	0.3	14735.4	0.00	0.00	4.48	1.70
20	5	0.4	17035.4	0.00	0.00	7.30	0.10
20	5	0.5	18941.0	0.00	0.54	3.39	4.27
20	5	0.6	20645.0	0.00	0.06	0.88	2.30
20	5	0.7	22269.5	0.02	0.00	3.49	4.43
20	5	0.8	23710.0	0.00	0.00	3.02	1.49
20	5	0.9	25082.5	0.00	0.00	1.98	0.24

Table B.19: % Error: Test Problem No. 9 (n=20)

n	p	α	Optimal Obj. Val.	% Error LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	5788.9	0.00	0.00	1.43	0.54
20	3	0.2	7056.8	0.00	0.00	2.17	1.63
20	3	0.3	8308.5	0.00	0.00	3.24	1.36
20	3	0.4	9264.8	0.00	0.00	5.62	6.48
20	3	0.5	10136.0	0.00	0.00	5.07	3.26
20	3	0.6	10904.8	0.00	0.00	7.74	6.86
20	3	0.7	11673.6	0.00	0.00	5.96	5.43
20	3	0.8	12440.2	0.00	0.00	4.50	4.50
20	3	0.9	13204.6	0.00	0.00	1.28	1.31
20	5	0.1	4626.8	0.00	0.00	0.00	1.08
20	5	0.2	6238.8	0.00	0.00	1.72	0.51
20	5	0.3	7548.5	0.00	0.00	7.72	0.80
20	5	0.4	8751.0	0.00	0.00	3.54	5.20
20	5	0.5	9875.5	0.27	0.00	4.10	4.89
20	5	0.6	10696.4	0.53	0.04	5.38	4.06
20	5	0.7	11517.3	0.37	0.37	1.11	4.71
20	5	0.8	12336.0	0.23	0.02	3.17	2.44
20	5	0.9	13152.5	0.00	0.11	1.66	2.39

Table B.20: % Error: Test Problem No. 10 (n=20)

B.1.3 Test Problems with 30 Demand Points

n	p	α	Optimal Obj. Val.	% Error LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	14564.0	0.00	0.00	13.68	8.35
30	3	0.4	18726.6	0.00	0.45	10.24	6.07
30	3	0.6	22364.6	0.00	0.73	7.21	3.65
30	3	0.8	25287.8	0.02	0.00	2.98	3.03
30	5	0.2	13250.6	0.00	0.00	5.88	4.95
30	5	0.4	17970.8	0.17	0.40	4.81	10.16
30	5	0.6	21800.0	0.96	0.85	4.13	7.09
30	5	0.8	24979.0	0.42	0.42	1.43	2.65

Table B.21: % Error: Test Problem No. 1 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	10617.8	0.00	0.00	1.50	16.87
30	3	0.4	13548.4	0.00	1.80	10.12	7.98
30	3	0.6	15525.4	0.00	0.00	6.08	6.08
30	3	0.8	17320.6	0.00	0.08	3.45	4.19
30	5	0.2	9488.6	0.00	0.18	5.84	12.37
30	5	0.4	12821.6	0.00	0.00	7.43	8.86
30	5	0.6	15199.8	0.80	0.00	1.87	6.25
30	5	0.8	17206.0	0.00	0.00	5.07	3.31

Table B.22: % Error: Test Problem No. 2 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	12763.6	0.00	1.70	7.52	7.40
30	3	0.4	15792.6	0.00	0.68	8.12	6.70
30	3	0.6	18162.8	0.00	0.00	8.43	5.22
30	3	0.8	20491.4	0.00	0.62	3.29	3.29
30	5	0.2	11317.6	0.00	0.00	11.75	8.43
30	5	0.4	15088.4	0.00	0.00	5.65	5.51
30	5	0.6	17948.4	0.00	0.00	3.82	5.18
30	5	0.8	20384.2	0.00	0.00	3.45	3.84

Table B.23: % Error: Test Problem No. 3 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	10881.2	0.00	0.28	8.03	14.47
30	3	0.4	13793.6	0.00	0.18	12.08	14.45
30	3	0.6	16222.6	0.00	3.16	8.54	9.25
30	3	0.8	18475.8	0.00	0.18	3.90	4.76
30	5	0.2	9131.8	0.00	0.00	12.02	17.89
30	5	0.4	12721.4	0.00	1.52	5.66	10.90
30	5	0.6	15900.2	0.59	0.59	2.64	4.00
30	5	0.8	18349.2	0.05	0.26	1.11	1.77

Table B.24: % Error: Test Problem No. 4 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	15202.8	0.00	0.88	4.24	21.25
30	3	0.4	18282.8	0.00	0.00	8.85	11.04
30	3	0.6	20653.8	0.00	0.14	8.56	7.31
30	3	0.8	22624.4	0.00	0.00	3.15	3.28
30	5	0.2	13387.6	0.00	0.00	3.62	7.56
30	5	0.4	17171.0	0.30	0.00	4.47	8.36
30	5	0.6	19978.2	0.00	0.00	3.84	8.34
30	5	0.8	22453.4	0.00	0.00	1.01	3.61

Table B.25: % Error: Test Problem No. 5 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	16442.8	0.00	0.00	2.98	2.95
30	3	0.4	22192.4	0.00	0.00	5.98	5.67
30	3	0.6	27278.2	0.00	0.91	3.18	3.28
30	3	0.8	31350.0	0.00	0.49	3.17	3.48
30	5	0.2	14619.0	0.00	0.49	4.23	7.04
30	5	0.4	20780.4	0.00	0.00	3.96	3.94
30	5	0.6	26625.4	0.47	0.47	2.42	4.01
30	5	0.8	31215.6	0.00	0.00	3.68	2.72

Table B.26: % Error: Test Problem No. 6 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	11419.8	0.00	1.83	14.28	12.49
30	3	0.4	13720.8	0.22	0.22	8.14	7.01
30	3	0.6	15349.0	0.00	0.00	4.39	4.39
30	3	0.8	16881.2	0.00	0.00	3.30	2.53
30	5	0.2	9989.4	0.00	0.00	14.54	14.69
30	5	0.4	12680.8	0.00	1.22	10.80	10.74
30	5	0.6	15071.4	0.67	0.00	3.87	6.43
30	5	0.8	16765.2	0.22	0.00	2.76	2.72

Table B.27: % Error: Test Problem No. 7 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	12155.8	0.00	0.00	6.78	9.05
30	3	0.4	14908.6	0.00	0.00	9.87	6.28
30	3	0.6	17563.8	0.00	0.05	6.50	6.19
30	3	0.8	19607.0	0.00	0.00	3.42	3.42
30	5	0.2	10809.6	0.00	0.83	5.00	5.45
30	5	0.4	14092.6	0.00	0.00	3.63	3.79
30	5	0.6	16966.8	0.00	0.19	5.08	3.02
30	5	0.8	19414.4	0.47	0.47	2.66	3.55

Table B.28: % Error: Test Problem No. 8 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	13081.2	1.31	0.00	0.88	2.06
30	3	0.4	14979.0	0.00	1.51	4.90	4.87
30	3	0.6	16660.0	0.00	1.03	8.44	9.07
30	3	0.8	18341.0	0.00	1.43	4.14	4.24
30	5	0.2	11672.4	0.00	0.00	1.33	3.65
30	5	0.4	14063.4	0.00	1.14	6.00	7.07
30	5	0.6	16157.6	0.00	0.17	6.22	7.86
30	5	0.8	18089.8	0.00	0.08	3.73	4.21

Table B.29: % Error: Test Problem No. 9 (n=30)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	15144.0	0.00	0.00	12.24	13.05
30	3	0.4	17548.6	0.00	0.00	16.22	14.29
30	3	0.6	19767.4	0.00	0.00	10.94	10.25
30	3	0.8	21738.4	0.00	0.00	4.79	3.14
30	5	0.2	14146.4	0.00	0.00	7.73	6.13
30	5	0.4	16800.4	1.85	1.55	5.37	14.01
30	5	0.6	19369.6	0.00	0.00	4.83	8.44
30	5	0.8	21543.0	0.00	0.00	3.86	3.72

Table B.30: % Error: Test Problem No. 10 (n=30)

B.1.4 Test Problems with 40 Demand Points

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	11185.2	0.00	0.00	11.17	9.06
40	3	0.4	14350.8	1.10	1.10	9.86	8.52
40	3	0.6	16959.2	0.00	0.17	6.46	5.52
40	3	0.8	19306.6	0.00	0.00	3.18	3.69

Table B.31: % Error: Test Problem No. 1 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	17038.6	0.00	2.67	4.96	11.99
40	3	0.4	19757.2	0.00	1.39	10.16	4.40
40	3	0.6	21927.2	0.00	1.51	13.49	12.57
40	3	0.8	24036.2	0.00	0.62	9.09	9.09

Table B.32: % Error: Test Problem No. 2 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	14109.0	0.00	0.00	18.39	16.20
40	3	0.4	16692.2	0.00	0.43	2.72	7.46
40	3	0.6	18679.4	0.00	0.78	9.41	9.41
40	3	0.8	20585.0	0.00	0.38	7.89	8.53

Table B.33: % Error: Test Problem No. 3 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	16048.4	0.00	1.27	5.89	15.25
40	3	0.4	19356.0	0.00	1.60	7.45	13.75
40	3	0.6	21884.4	0.00	0.00	13.02	12.62
40	3	0.8	24253.8	0.05	0.05	5.00	7.01

Table B.34: % Error: Test Problem No. 4 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	17218.6	0.00	1.19	11.89	11.19
40	3	0.4	22006.0	0.00	0.00	6.37	7.60
40	3	0.6	26380.0	0.00	0.00	6.07	6.09
40	3	0.8	30750.0	0.00	0.00	2.63	2.96

Table B.35: % Error: Test Problem No. 5 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	14259.4	0.00	1.21	11.61	10.53
40	3	0.4	16769.8	0.00	3.67	11.54	11.54
40	3	0.6	19280.2	0.00	0.00	10.64	6.52
40	3	0.8	21790.6	0.00	0.00	2.87	4.07

Table B.36: % Error: Test Problem No. 6 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	18077.2	0.00	0.00	11.97	9.26
40	3	0.4	24083.2	0.00	0.90	7.38	6.63
40	3	0.6	29848.0	0.00	0.18	4.31	5.03
40	3	0.8	35241.2	0.00	0.31	2.96	3.49

Table B.37: % Error: Test Problem No. 7 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	15761.4	0.00	0.09	5.69	13.74
40	3	0.4	17451.0	0.00	0.00	16.31	14.27
40	3	0.6	19021.0	0.00	0.00	13.49	12.37
40	3	0.8	20550.0	0.00	0.92	9.04	8.74

Table B.38: % Error: Test Problem No. 8 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	13515.4	0.00	0.00	6.30	6.61
40	3	0.4	16612.6	0.00	0.01	5.62	9.93
40	3	0.6	18628.4	0.00	0.00	9.39	9.58
40	3	0.8	20594.0	0.00	0.00	5.25	4.47

Table B.39: % Error: Test Problem No. 9 (n=40)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	12738.0	0.00	0.00	8.56	17.20
40	3	0.4	16445.0	0.00	0.00	8.55	5.01
40	3	0.6	19538.6	0.00	0.00	8.99	8.01
40	3	0.8	22368.8	0.00	1.15	5.14	4.60

Table B.40: % Error For Test Problems 10 (n=40)

B.1.5 Test Problems with 50 Demand Points

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	18506.6	0.00	0.00	4.14	12.49
50	3	0.4	22514.2	0.00	0.70	13.39	14.55
50	3	0.6	25540.0	0.00	0.10	9.15	9.62
50	3	0.8	27871.2	0.00	0.00	7.19	7.06

Table B.41: % Error: Test Problem No. 1 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	20555.4	0.00	0.00	11.08	15.74
50	3	0.4	23454.4	0.00	0.00	15.93	14.22
50	3	0.6	26239.2	0.00	0.00	9.77	9.77
50	3	0.8	28808.6	0.00	0.00	4.05	5.64

Table B.42: % Error: Test Problem No. 2 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	13562.6	0.00	0.00	17.30	11.33
50	3	0.4	16674.0	0.00	0.00	8.13	6.71
50	3	0.6	18889.4	0.00	1.07	6.79	6.19
50	3	0.8	20753.6	0.00	0.00	3.63	3.10

Table B.43: % Error: Test Problem No. 3 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23798.0	0.00	0.75	15.08	16.53
50	3	0.4	27484.2	0.00	0.00	13.53	11.81
50	3	0.6	30525.6	0.00	0.00	7.38	6.96
50	3	0.8	33334.8	0.00	0.00	3.79	3.29

Table B.44: % Error: Test Problem No. 4 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	15526.2	0.00	1.23	17.07	14.97
50	3	0.4	18029.2	0.00	0.92	9.83	10.27
50	3	0.6	20092.8	0.00	1.62	9.79	9.79
50	3	0.8	22060.0	0.00	0.62	4.60	5.71

Table B.45: % Error: Test Problem No. 5 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	19683.2	0.00	0.00	15.63	15.63
50	3	0.4	24672.6	1.29	0.00	7.49	5.23
50	3	0.6	29002.8	0.00	1.02	7.20	13.00
50	3	0.8	31840.2	0.00	0.30	7.74	7.07

Table B.46: % Error: Test Problem No. 6 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	12286.0	0.00	0.60	17.55	15.92
50	3	0.4	14964.4	0.00	0.00	9.10	12.53
50	3	0.6	17415.4	0.00	0.00	6.08	6.08
50	3	0.8	19710.2	0.00	0.00	2.33	1.54

Table B.47: % Error: Test Problem No. 7 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	18422.6	0.00	0.18	4.38	6.44
50	3	0.4	21738.8	0.00	1.66	7.32	12.65
50	3	0.6	24257.4	0.00	1.58	8.42	8.81
50	3	0.8	26442.8	0.00	0.68	4.76	4.76

Table B.48: % Error: Test Problem No. 8 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	18082.8	0.00	0.00	10.64	13.28
50	3	0.4	21894.6	0.00	2.64	9.88	12.44
50	3	0.6	25598.4	0.00	1.36	6.61	9.12
50	3	0.8	29086.8	0.00	0.61	4.85	4.54

Table B.49: % Error: Test Problem No. 9 (n=50)

n	p	α	Optimal Obj. Val.	% Error			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	19593.4	0.00	0.00	7.21	13.28
50	3	0.4	24058.4	0.00	0.00	11.57	14.54
50	3	0.6	28326.0	0.00	0.00	12.37	11.80
50	3	0.8	32408.8	0.00	0.00	5.89	4.68

Table B.50: % Error: Test Problem No. 10 (n=50)

B.2 Comparisons for Computation Times

This section gives the performance of heuristics with respect to computation times. In all cases LRA type heuristics took less time than ARL type heuristic. However, this is expected as discussed in Chapter 4, because at each iteration of the ARL type heuristics determining the locations take $O(pn^2)$ time, where as at each iteration of LRA type heuristics allocations are calculated in $O(pn)$ time. Because of the effect of n in the computation time ARL type heuristics took even longer than the B&B algorithm for relatively large n and small p values. However, of course, B&B algorithm is an exponential time algorithm and as n and p increases, the computation time increase very rapidly. Therefore. for example, for $n = 30$ and $p = 5$ B&B algorithm became much slower than ARL type heuristics.

B.2.1 Test Problems with 10 Demand Points

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	0	0
10	3	0.2	0	0	0	0	1
10	3	0.3	0	0	0	1	0
10	3	0.4	0	0	0	0	1
10	3	0.5	0	0	0	1	0
10	3	0.6	0	0	0	0	1
10	3	0.7	0	0	0	0	0
10	3	0.8	0	0	0	1	0
10	3	0.9	0	0	0	0	1
10	5	0.1	2	0	1	5	8
10	5	0.2	2	1	0	5	7
10	5	0.3	0	0	0	3	8
10	5	0.4	1	1	1	4	8
10	5	0.5	1	0	0	3	8
10	5	0.6	0	0	1	3	6
10	5	0.7	1	1	0	2	6
10	5	0.8	0	0	0	4	6
10	5	0.9	0	0	1	5	6

Table B.51: Processing Times: Test Problem No. 1 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	1	0	0
10	3	0.2	0	1	0	1	1
10	3	0.3	0	0	0	0	1
10	3	0.4	0	0	0	0	0
10	3	0.5	0	0	0	0	1
10	3	0.6	0	0	0	1	0
10	3	0.7	0	0	0	0	0
10	3	0.8	0	0	0	0	1
10	3	0.9	0	0	0	0	0
10	5	0.1	2	1	0	3	7
10	5	0.2	2	0	1	3	6
10	5	0.3	2	0	0	4	5
10	5	0.4	1	0	1	4	6
10	5	0.5	1	1	0	5	6
10	5	0.6	1	0	0	6	6
10	5	0.7	1	0	1	6	7
10	5	0.8	0	0	0	5	6
10	5	0.9	1	0	1	4	6

Table B.52: Processing Times: Test Problem No. 2 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	1	1	0
10	3	0.2	0	0	0	0	0
10	3	0.3	0	0	0	0	0
10	3	0.4	0	0	0	1	1
10	3	0.5	0	1	0	0	0
10	3	0.6	0	0	0	0	1
10	3	0.7	0	0	0	1	0
10	3	0.8	0	0	0	0	0
10	3	0.9	0	0	0	0	1
10	5	0.1	2	0	0	5	8
10	5	0.2	1	0	1	7	8
10	5	0.3	1	1	0	5	8
10	5	0.4	1	0	1	7	6
10	5	0.5	1	0	0	4	8
10	5	0.6	0	1	0	6	5
10	5	0.7	1	0	1	6	6
10	5	0.8	0	0	0	3	6
10	5	0.9	1	1	1	5	6

Table B.53: Processing Times: Test Problem No. 3 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	1	0
10	3	0.2	0	0	1	0	1
10	3	0.3	0	0	0	1	1
10	3	0.4	0	0	0	0	0
10	3	0.5	0	0	0	0	0
10	3	0.6	0	0	0	1	1
10	3	0.7	0	0	0	0	0
10	3	0.8	0	1	0	0	0
10	3	0.9	0	0	0	0	1
10	5	0.1	1	0	0	4	8
10	5	0.2	2	0	0	5	7
10	5	0.3	0	1	1	7	7
10	5	0.4	1	0	0	8	7
10	5	0.5	0	1	1	7	6
10	5	0.6	1	0	0	4	6
10	5	0.7	0	0	1	7	6
10	5	0.8	0	1	0	7	7
10	5	0.9	1	0	1	7	6

Table B.54: Processing Times: Test Problem No. 4 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	1	0	0	1	0
10	3	0.2	0	0	0	0	1
10	3	0.3	0	0	1	0	0
10	3	0.4	0	0	0	1	0
10	3	0.5	0	0	0	0	1
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	0	0	1
10	3	0.8	0	0	0	1	0
10	3	0.9	0	1	0	0	0
10	5	0.1	1	0	0	5	7
10	5	0.2	1	1	1	7	7
10	5	0.3	1	0	0	5	5
10	5	0.4	1	1	1	6	6
10	5	0.5	1	0	0	5	6
10	5	0.6	1	1	1	5	6
10	5	0.7	0	0	0	5	6
10	5	0.8	1	1	1	4	7
10	5	0.9	0	0	0	4	5

Table B.55: Processing Times: Test Problem No. 5 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	0	1
10	3	0.2	0	0	0	1	0
10	3	0.3	0	0	0	0	0
10	3	0.4	0	0	1	0	0
10	3	0.5	0	0	0	1	1
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	0	0	0
10	3	0.8	0	0	0	1	1
10	3	0.9	0	1	0	0	0
10	5	0.1	2	1	1	5	8
10	5	0.2	2	0	0	5	7
10	5	0.3	1	0	1	5	6
10	5	0.4	1	1	0	6	7
10	5	0.5	1	0	1	6	7
10	5	0.6	0	0	0	5	6
10	5	0.7	0	1	0	6	6
10	5	0.8	1	0	1	6	6
10	5	0.9	0	0	0	5	5

Table B.56: Processing Times: Test Problem No. 6 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	1	1
10	3	0.2	0	0	0	0	0
10	3	0.3	0	0	0	0	1
10	3	0.4	0	0	1	1	0
10	3	0.5	0	0	0	0	1
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	0	1	0
10	3	0.8	0	0	0	0	1
10	3	0.9	0	0	0	0	0
10	5	0.1	2	1	1	5	6
10	5	0.2	1	0	0	5	8
10	5	0.3	2	0	0	7	6
10	5	0.4	1	1	1	6	7
10	5	0.5	1	0	0	5	6
10	5	0.6	0	0	1	6	6
10	5	0.7	1	0	0	5	6
10	5	0.8	0	1	1	7	7
10	5	0.9	0	0	0	6	7

Table B.57: Processing Times: Test Problem No. 7 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	1	1
10	3	0.2	0	1	0	0	1
10	3	0.3	0	0	0	1	0
10	3	0.4	0	0	0	1	1
10	3	0.5	0	0	0	0	0
10	3	0.6	0	0	1	0	1
10	3	0.7	0	0	0	1	0
10	3	0.8	0	0	0	0	1
10	3	0.9	0	0	0	0	0
10	5	0.1	3	0	0	5	7
10	5	0.2	1	1	1	5	9
10	5	0.3	1	0	0	5	6
10	5	0.4	1	1	1	5	5
10	5	0.5	0	0	0	5	6
10	5	0.6	1	0	0	5	4
10	5	0.7	0	1	1	6	5
10	5	0.8	0	0	0	6	4
10	5	0.9	0	0	1	6	8

Table B.58: Processing Times: Test Problem No. 8 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	1	1
10	3	0.2	0	0	0	0	0
10	3	0.3	0	0	0	0	1
10	3	0.4	0	1	0	1	0
10	3	0.5	0	0	0	0	1
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	1	1	0
10	3	0.8	0	0	0	0	1
10	3	0.9	0	0	0	0	0
10	5	0.1	3	1	0	5	7
10	5	0.2	2	0	1	5	6
10	5	0.3	2	1	0	5	5
10	5	0.4	1	0	1	6	5
10	5	0.5	1	0	0	6	6
10	5	0.6	1	1	0	5	6
10	5	0.7	0	0	1	6	5
10	5	0.8	1	1	0	5	5
10	5	0.9	0	0	1	5	6

Table B.59: Processing Times: Test Problem No. 9 (n=10)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
10	3	0.1	0	0	0	1	1
10	3	0.2	0	0	0	0	0
10	3	0.3	0	0	0	1	1
10	3	0.4	0	1	0	0	0
10	3	0.5	0	0	0	0	1
10	3	0.6	0	0	0	0	0
10	3	0.7	0	0	1	0	1
10	3	0.8	0	0	0	0	0
10	3	0.9	0	0	0	0	0
10	5	0.1	2	1	0	6	9
10	5	0.2	1	0	1	5	7
10	5	0.3	1	1	0	6	7
10	5	0.4	1	0	1	8	6
10	5	0.5	1	0	0	6	7
10	5	0.6	1	1	1	8	4
10	5	0.7	0	0	0	6	5
10	5	0.8	1	1	1	5	6
10	5	0.9	0	0	0	6	6

Table B.60: Processing Times: Test Problem No. 10 (n=10)

B.2.2 Test Problems with 20 Demand Points

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	0	0	4	6
20	3	0.2	1	1	1	3	5
20	3	0.3	0	1	0	4	7
20	3	0.4	0	0	1	3	4
20	3	0.5	0	1	0	2	3
20	3	0.6	1	1	1	3	4
20	3	0.7	0	0	0	2	2
20	3	0.8	0	1	1	2	3
20	3	0.9	0	0	0	2	3
20	5	0.1	182	3	3	51	74
20	5	0.2	144	4	4	58	63
20	5	0.3	95	4	2	66	53
20	5	0.4	79	4	3	53	45
20	5	0.5	67	4	3	31	41
20	5	0.6	34	5	3	43	38
20	5	0.7	33	4	2	35	41
20	5	0.8	23	5	3	37	39
20	5	0.9	23	4	3	34	38

Table B.61: Processing Times: Test Problem No. 1 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	1	1	4	6
20	3	0.2	1	0	0	4	3
20	3	0.3	1	0	1	3	4
20	3	0.4	0	1	0	1	3
20	3	0.5	0	0	1	3	3
20	3	0.6	0	1	0	2	2
20	3	0.7	1	0	1	2	3
20	3	0.8	0	1	0	1	2
20	3	0.9	0	0	1	2	3
20	5	0.1	226	4	3	65	87
20	5	0.2	184	4	4	62	80
20	5	0.3	123	4	3	48	64
20	5	0.4	94	4	3	62	43
20	5	0.5	87	3	2	57	53
20	5	0.6	63	3	3	30	45
20	5	0.7	34	3	2	36	40
20	5	0.8	33	4	3	37	38
20	5	0.9	11	2	2	25	46

Table B.62: Processing Times: Test Problem No. 2 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	1	0	9	7
20	3	0.2	1	0	1	5	4
20	3	0.3	0	1	0	5	4
20	3	0.4	1	0	1	2	4
20	3	0.5	0	0	0	3	3
20	3	0.6	0	1	1	2	2
20	3	0.7	0	0	0	1	3
20	3	0.8	0	1	0	2	2
20	3	0.9	0	0	1	3	4
20	5	0.1	232	3	3	47	81
20	5	0.2	160	4	3	58	66
20	5	0.3	94	4	3	79	53
20	5	0.4	70	5	2	53	56
20	5	0.5	53	5	3	75	65
20	5	0.6	32	4	2	66	53
20	5	0.7	22	3	2	74	53
20	5	0.8	12	2	2	75	58
20	5	0.9	12	2	2	63	52

Table B.63: Processing Times: Test Problem No. 3 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	1	1	0	3	6
20	3	0.2	0	0	1	3	5
20	3	0.3	1	1	0	3	4
20	3	0.4	0	0	1	5	5
20	3	0.5	1	1	0	6	3
20	3	0.6	0	0	1	4	4
20	3	0.7	0	1	0	2	4
20	3	0.8	0	0	1	2	2
20	3	0.9	0	0	0	3	4
20	5	0.1	215	4	4	36	87
20	5	0.2	191	3	3	37	63
20	5	0.3	120	4	3	66	74
20	5	0.4	107	3	2	64	63
20	5	0.5	89	3	2	78	52
20	5	0.6	64	2	2	53	56
20	5	0.7	44	3	3	57	45
20	5	0.8	22	3	2	56	46
20	5	0.9	12	2	3	53	56

Table B.64: Processing Times: Test Problem No. 4 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	1	1	1	3	6
20	3	0.2	0	0	0	4	3
20	3	0.3	1	0	1	3	2
20	3	0.4	0	1	0	2	2
20	3	0.5	0	0	1	2	3
20	3	0.6	0	0	0	2	2
20	3	0.7	0	1	1	1	2
20	3	0.8	1	0	0	2	2
20	3	0.9	0	0	0	2	2
20	5	0.1	208	4	3	31	83
20	5	0.2	173	3	4	41	72
20	5	0.3	144	4	3	87	58
20	5	0.4	118	3	2	67	57
20	5	0.5	72	2	2	76	50
20	5	0.6	33	2	2	85	55
20	5	0.7	11	2	2	68	54
20	5	0.8	13	3	3	70	65
20	5	0.9	11	2	2	78	75

Table B.65: Processing Times: Test Problem No. 5 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	1	1	10	10
20	3	0.2	1	1	1	4	7
20	3	0.3	0	0	0	3	5
20	3	0.4	1	1	1	2	5
20	3	0.5	0	1	0	2	3
20	3	0.6	0	0	1	2	3
20	3	0.7	0	1	1	3	3
20	3	0.8	0	1	0	1	2
20	3	0.9	0	0	1	2	2
20	5	0.1	220	4	4	40	94
20	5	0.2	182	4	3	48	66
20	5	0.3	156	4	4	29	52
20	5	0.4	117	3	2	38	48
20	5	0.5	64	3	2	52	47
20	5	0.6	46	4	2	56	40
20	5	0.7	45	3	3	59	36
20	5	0.8	34	2	2	38	42
20	5	0.9	22	2	2	47	35

Table B.66: Processing Times: Test Problem No. 6 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	1	1	0	3	6
20	3	0.2	1	0	1	2	5
20	3	0.3	0	1	0	4	3
20	3	0.4	0	0	1	3	4
20	3	0.5	1	1	0	6	4
20	3	0.6	0	0	1	2	3
20	3	0.7	0	1	0	1	3
20	3	0.8	0	1	1	2	3
20	3	0.9	0	0	1	3	3
20	5	0.1	230	3	5	53	74
20	5	0.2	175	4	3	41	50
20	5	0.3	117	2	3	32	60
20	5	0.4	67	3	3	74	68
20	5	0.5	61	2	3	57	58
20	5	0.6	57	3	4	74	54
20	5	0.7	32	3	3	56	44
20	5	0.8	23	3	3	46	49
20	5	0.9	22	4	4	50	59

Table B.67: Processing Times: Test Problem No. 7 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	1	1	0	3	11
20	3	0.2	0	0	1	3	6
20	3	0.3	1	1	0	3	5
20	3	0.4	0	0	1	2	3
20	3	0.5	0	1	0	1	3
20	3	0.6	0	0	1	4	3
20	3	0.7	0	1	0	2	3
20	3	0.8	1	0	0	2	3
20	3	0.9	0	1	1	2	2
20	5	0.1	211	3	3	35	76
20	5	0.2	143	5	3	32	61
20	5	0.3	112	4	4	28	51
20	5	0.4	88	3	4	35	56
20	5	0.5	78	2	2	34	45
20	5	0.6	68	5	3	37	48
20	5	0.7	45	3	3	53	48
20	5	0.8	46	2	2	48	46
20	5	0.9	22	2	2	44	46

Table B.68: Processing Times: Test Problem No. 8 (n=20)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	0	0	4	5
20	3	0.2	1	1	1	4	4
20	3	0.3	0	0	0	2	3
20	3	0.4	1	1	1	2	2
20	3	0.5	0	0	0	1	2
20	3	0.6	0	1	0	2	3
20	3	0.7	1	0	1	1	2
20	3	0.8	0	0	0	4	2
20	3	0.9	0	0	1	1	3
20	5	0.1	210	2	3	34	65
20	5	0.2	165	3	3	42	71
20	5	0.3	143	3	3	48	49
20	5	0.4	137	2	3	64	52
20	5	0.5	103	2	3	57	47
20	5	0.6	67	3	2	57	57
20	5	0.7	61	2	3	76	43
20	5	0.8	41	2	2	64	56
20	5	0.9	11	2	3	52	53

Table B.69: Processing Times: Test Problem No. 9 (n=20)

n	p	α	CPU Times (in seconds) B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	3	0.1	0	0	0	3	10
20	3	0.2	1	0	1	3	6
20	3	0.3	0	1	0	2	5
20	3	0.4	1	0	1	2	4
20	3	0.5	0	1	0	2	4
20	3	0.6	0	0	1	2	3
20	3	0.7	0	1	1	3	4
20	3	0.8	0	0	0	3	3
20	3	0.9	0	1	0	5	4
20	5	0.1	202	4	4	42	94
20	5	0.2	132	3	3	59	64
20	5	0.3	112	4	4	49	71
20	5	0.4	108	3	3	55	46
20	5	0.5	102	2	4	48	43
20	5	0.6	68	3	4	49	45
20	5	0.7	55	2	3	36	33
20	5	0.8	33	3	4	60	32
20	5	0.9	11	2	3	49	39

Table B.70: Processing Times: Test Problem No. 10 (n=20)

B.2.3 Test Problems with 30 Demand Points

n	p	α	CPU Times (in seconds) B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	10	28
30	3	0.4	1	1	1	7	21
30	3	0.6	1	1	1	7	13
30	3	0.8	0	2	1	5	9
30	5	0.2	2431	9	9	112	326
30	5	0.4	1016	7	10	213	241
30	5	0.6	573	10	7	152	130
30	5	0.8	285	7	6	192	157

Table B.71: Processing Times: Test Problem No. 1 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	2	1	29	29
30	3	0.4	1	1	1	10	11
30	3	0.6	1	1	1	5	9
30	3	0.8	0	1	1	6	10
30	5	0.2	2074	13	8	292	270
30	5	0.4	823	8	6	305	164
30	5	0.6	229	7	5	286	151
30	5	0.8	191	11	8	190	177

Table B.72: Processing Times: Test Problem No. 2 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	15	24
30	3	0.4	1	1	1	6	13
30	3	0.6	1	1	1	6	11
30	3	0.8	0	2	2	11	13
30	5	0.2	2329	11	10	81	235
30	5	0.4	1594	9	7	206	208
30	5	0.6	359	6	5	239	145
30	5	0.8	193	6	5	191	181

Table B.73: Processing Times: Test Problem No. 3 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	16	18
30	3	0.4	2	1	1	9	10
30	3	0.6	0	1	2	10	16
30	3	0.8	1	2	1	8	8
30	5	0.2	2000	10	9	204	235
30	5	0.4	1142	9	11	310	185
30	5	0.6	720	6	5	264	180
30	5	0.8	194	8	9	318	212

Table B.74: Processing Times: Test Problem No. 4 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	2	1	1	21	12
30	3	0.4	1	2	1	12	12
30	3	0.6	1	2	1	9	10
30	3	0.8	0	1	1	29	14
30	5	0.2	2476	10	9	501	305
30	5	0.4	1411	9	9	503	254
30	5	0.6	499	10	5	383	216
30	5	0.8	279	8	5	261	250

Table B.75: Processing Times: Test Problem No. 5 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	2	1	28	36
30	3	0.4	2	1	2	15	17
30	3	0.6	1	1	1	34	18
30	3	0.8	0	2	1	15	18
30	5	0.2	2364	11	7	272	319
30	5	0.4	1162	10	8	268	280
30	5	0.6	675	7	7	298	180
30	5	0.8	191	6	6	177	200

Table B.76: Processing Times: Test Problem No. 6 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	7	18
30	3	0.4	1	1	1	7	12
30	3	0.6	0	1	1	5	8
30	3	0.8	1	2	1	6	7
30	5	0.2	1900	10	10	380	238
30	5	0.4	854	8	10	287	187
30	5	0.6	481	9	8	305	102
30	5	0.8	192	8	9	353	163

Table B.77: Processing Times: Test Problem No. 7 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	8	14
30	3	0.4	2	2	2	14	18
30	3	0.6	1	1	1	11	15
30	3	0.8	0	1	1	11	11
30	5	0.2	2316	14	11	250	285
30	5	0.4	1570	12	8	142	129
30	5	0.6	869	13	7	166	157
30	5	0.8	288	6	5	236	145

Table B.78: Processing Times: Test Problem No. 8 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	2	1	12	34
30	3	0.4	1	2	1	12	18
30	3	0.6	1	2	2	7	8
30	3	0.8	0	2	1	7	9
30	5	0.2	2116	17	9	197	261
30	5	0.4	1123	14	7	418	225
30	5	0.6	385	12	7	291	143
30	5	0.8	291	12	7	176	143

Table B.79: Processing Times: Test Problem No. 9 (n=30)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	3	0.2	3	1	1	14	1C
30	3	0.4	1	1	1	14	13
30	3	0.6	1	1	1	7	11
30	3	0.8	0	1	1	21	15
30	5	0.2	2225	10	10	242	299
30	5	0.4	806	5	5	274	147
30	5	0.6	289	6	8	231	144
30	5	0.8	187	7	7	275	164

Table B.80: Processing Times: Test Problem No. 10 (n=30)

B.2.4 Test Problems with 40 Demand Points

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	8	2	4	31	50
40	3	0.4	5	2	7	20	39
40	3	0.6	1	4	5	17	32
40	3	0.8	1	3	5	24	37

Table B.81: Processing Times: Test Problem No. 1 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	10	3	6	90	62
40	3	0.4	3	2	5	46	56
40	3	0.6	2	3	4	33	34
40	3	0.8	0	3	4	17	21

Table B.82: Processing Times: Test Problem No. 2 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	10	3	7	32	35
40	3	0.4	3	3	6	52	40
40	3	0.6	2	3	3	22	21
40	3	0.8	0	3	3	16	25

Table B.83: Processing Times: Test Problem No. 3 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	3	6	70	60
40	3	0.4	7	4	5	39	33
40	3	0.6	2	3	5	16	19
40	3	0.8	1	3	5	34	26

Table B.84: Processing Times: Test Problem No. 4 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	3	5	24	46
40	3	0.4	4	2	4	48	49
40	3	0.6	2	2	3	18	28
40	3	0.8	0	2	4	17	23

Table B.85: Processing Times: Test Problem No. 5 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	10	3	6	23	57
40	3	0.4	5	4	6	15	27
40	3	0.6	2	4	4	24	28
40	3	0.8	1	5	4	21	28

Table B.86: Processing Times: Test Problem No. 6 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	3	10	22	48
40	3	0.4	4	3	7	29	38
40	3	0.6	3	4	5	22	43
40	3	0.8	1	4	5	24	35

Table B.87: Processing Times: Test Problem No. 7 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	3	4	58	46
40	3	0.4	4	2	6	15	24
40	3	0.6	1	3	5	20	27
40	3	0.8	0	2	5	16	26

Table B.88: Processing Times: Test Problem No. 8 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	2	7	108	86
40	3	0.4	5	2	5	35	27
40	3	0.6	2	2	6	21	24
40	3	0.8	1	2	5	20	27

Table B.89: Processing Times: Test Problem No. 9 (n=40)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	3	0.2	9	2	5	20	34
40	3	0.4	5	2	7	21	47
40	3	0.6	2	3	5	21	29
40	3	0.8	1	3	3	18	35

Table B.90: Processing Times: Test Problem No. 10 (n=40)

B.2.5 Test Problems with 50 Demand Points

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23	6	3	219	136
50	3	0.4	15	5	6	48	66
50	3	0.6	5	6	3	45	74
50	3	0.8	3	5	4	50	73

Table B.91: Processing Times: Test Problem No. 1 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	22	6	3	110	94
50	3	0.4	11	7	3	46	72
50	3	0.6	4	7	3	38	61
50	3	0.8	1	6	2	57	68

Table B.92: Processing Times: Test Problem No. 2 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23	5	4	79	148
50	3	0.4	10	6	4	55	101
50	3	0.6	5	7	4	37	64
50	3	0.8	1	5	2	40	66

Table B.93: Processing Times: Test Problem No. 3 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23	4	3	159	77
50	3	0.4	11	5	4	38	79
50	3	0.6	3	3	4	72	67
50	3	0.8	1	5	3	78	81

Table B.94: Processing Times: Test Problem No. 4 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23	6	4	53	82
50	3	0.4	8	7	4	150	109
50	3	0.6	3	6	2	49	75
50	3	0.8	1	5	3	58	66

Table B.95: Processing Times: Test Problem No. 5 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	24	7	4	51	91
50	3	0.4	15	4	4	137	135
50	3	0.6	2	6	2	64	49
50	3	0.8	1	4	3	39	64

Table B.96: Processing Times: Test Problem No. 6 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	20	5	3	44	95
50	3	0.4	12	4	3	104	69
50	3	0.6	3	4	4	36	61
50	3	0.8	2	4	4	89	96

Table B.97: Processing Times: Test Problem No. 7 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	22	4	4	116	156
50	3	0.4	13	4	4	112	72
50	3	0.6	4	5	3	100	64
50	3	0.8	2	5	4	37	59

Table B.98: Processing Times: Test Problem No. 8 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	24	6	5	47	74
50	3	0.4	12	5	3	46	63
50	3	0.6	2	4	3	51	82
50	3	0.8	1	5	3	53	63

Table B.99: Processing Times: Test Problem No. 9 (n=50)

n	p	α	CPU Times (in seconds)				
			B&B	LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	3	0.2	23	5	4	329	160
50	3	0.4	17	6	5	107	96
50	3	0.6	5	5	2	53	76
50	3	0.8	1	6	3	49	63

Table B.100: Processing Times: Test Problem No. 10 (n=50)

Appendix C

Comparisons for Each Test Problem: Heuristic vs Heuristic

In this appendix, we compare the heuristics between themselves for larger problem sizes where the optimum solutions were not available. Section C.1 reports comparisons with respect to the best solution found among all heuristics. Therefore, although the percentage difference of a given heuristic solution from the best solution found gives a basis for comparison for heuristics between themselves, it does not provide immediate information with respect to the closeness of the results to the actual optimum. Section C.2 gives the computation times of each heuristic algorithm for each of the test problems.

C.1 Comparisons for Closeness the Best Solution Found

In all the testing done in this section, LRA type heuristics again performed much better compared to ARL type heuristics. In almost all cases, one of the LRA type heuristics was the one the find the best solution among all four of the algorithms.

Note here that a value of 0.00 for % Difference states that the best solution was found by the given algorithm. Compared to the results in Section B.1, the results for LRA(B) seemed to improve a bit, that is, LRA(B) found solutions that are as good as or better than LRA(A) more often.

C.1.1 Test Problems with 20 Demand Points

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	5172.1	0.00	0.00	0.67	0.95
20	7	0.2	8159.6	0.00	0.11	0.32	1.10
20	7	0.3	10883.0	0.87	0.00	2.77	1.15
20	7	0.4	13508.8	1.25	0.00	1.31	0.68
20	7	0.5	16017.5	0.63	0.00	1.54	0.24
20	7	0.6	18299.6	0.00	0.00	2.16	2.75
20	7	0.7	20818.1	0.00	0.00	1.11	1.19
20	7	0.8	23116.4	0.25	0.25	1.70	0.00
20	7	0.9	25326.6	0.00	0.00	1.44	0.79

Table C.1: % Difference: Test Problem No. 1 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	6897.0	0.00	0.00	3.09	2.70
20	7	0.2	9801.6	0.00	0.00	0.94	1.41
20	7	0.3	12189.0	0.00	0.00	1.77	1.93
20	7	0.4	14155.0	0.28	0.72	0.94	0.00
20	7	0.5	16209.0	0.00	0.21	1.32	5.83
20	7	0.6	18030.2	0.00	0.00	6.12	6.42
20	7	0.7	19743.5	0.00	0.00	2.72	1.79
20	7	0.8	21410.0	0.00	0.00	6.13	1.31
20	7	0.9	22966.9	0.00	0.00	3.65	4.71

Table C.2: % Difference: Test Problem No. 2 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	7202.9	0.00	0.36	0.85	1.39
20	7	0.2	9727.2	0.00	0.00	0.81	3.42
20	7	0.3	11743.4	0.00	0.00	6.87	14.35
20	7	0.4	13118.2	0.00	0.00	7.60	6.40
20	7	0.5	14438.0	0.00	0.00	10.80	9.98
20	7	0.6	15613.4	0.00	0.00	2.70	10.26
20	7	0.7	16699.8	0.00	0.00	7.86	7.90
20	7	0.8	17666.0	0.00	0.00	7.75	12.56
20	7	0.9	18402.5	0.00	0.00	8.43	13.01

Table C.3: % Difference: Test Problem No. 3 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	5857.2	0.00	0.00	0.81	2.95
20	7	0.2	9207.2	0.00	0.00	3.47	3.09
20	7	0.3	11670.6	1.30	0.00	2.33	4.45
20	7	0.4	13848.4	0.00	0.00	0.34	1.42
20	7	0.5	15821.0	0.00	0.00	0.35	3.68
20	7	0.6	17755.2	0.00	0.00	0.92	2.53
20	7	0.7	19689.4	0.00	0.00	0.46	1.66
20	7	0.8	21544.4	0.37	0.00	2.12	2.00
20	7	0.9	23060.3	0.12	0.00	0.92	0.95

Table C.4: % Difference: Test Problem No. 4 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	6582.9	0.00	0.00	0.90	0.85
20	7	0.2	8649.4	0.00	0.00	2.78	0.58
20	7	0.3	10281.1	0.00	0.04	0.18	0.83
20	7	0.4	11179.2	0.00	0.00	2.14	1.86
20	7	0.5	11863.0	0.00	0.00	4.81	5.83
20	7	0.6	12431.6	0.00	0.00	7.04	6.84
20	7	0.7	13000.2	0.00	0.00	10.52	5.07
20	7	0.8	13568.8	0.00	0.00	6.24	4.62
20	7	0.9	14131.0	0.00	0.00	3.81	2.94

Table C.5: % Difference: Test Problem No. 5 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	5447.9	1.16	0.00	2.02	2.61
20	7	0.2	7916.6	0.00	0.00	2.10	7.85
20	7	0.3	10179.4	0.00	0.00	1.70	0.84
20	7	0.4	11708.0	0.35	0.35	1.48	0.00
20	7	0.5	12763.5	1.00	0.00	1.98	3.06
20	7	0.6	13936.8	0.00	0.00	2.22	3.16
20	7	0.7	14915.5	0.51	0.00	1.68	2.35
20	7	0.8	15899.0	0.32	0.00	0.84	1.67
20	7	0.9	16908.1	0.00	0.00	1.44	2.50

Table C.6: % Difference: Test Problem No. 6 (n=20)

n	p	α	Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	7600.9	0.00	0.00	3.24	2.97
20	7	0.2	10024.4	0.00	0.00	3.88	1.68
20	7	0.3	11466.7	1.95	0.00	0.87	0.49
20	7	0.4	12545.6	0.00	0.00	0.27	0.83
20	7	0.5	13311.5	0.00	0.35	0.74	1.74
20	7	0.6	13873.2	0.13	0.00	0.12	1.33
20	7	0.7	14422.3	0.00	0.00	0.76	0.47
20	7	0.8	14936.2	0.00	0.00	1.04	8.60
20	7	0.9	15450.1	0.00	0.00	1.19	2.96

Table C.7: % Difference: Test Problem No. 7 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	6161.0	0.00	0.00	2.17	0.28
20	7	0.2	8512.0	4.67	0.00	0.00	5.96
20	7	0.3	10668.1	0.00	0.00	9.95	4.32
20	7	0.4	12736.4	1.90	1.50	0.14	0.00
20	7	0.5	14485.0	1.34	0.00	3.03	4.84
20	7	0.6	16001.2	0.00	0.00	5.91	1.56
20	7	0.7	17386.9	0.35	0.35	0.01	0.00
20	7	0.8	18744.6	0.00	0.00	3.72	3.49
20	7	0.9	19983.8	0.00	0.00	0.05	1.33

Table C.8: % Difference: Test Problem No. 8 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	7536.0	0.00	0.00	5.68	2.80
20	7	0.2	11141.0	0.00	0.40	9.57	3.53
20	7	0.3	14319.9	0.00	1.17	4.20	6.48
20	7	0.4	16946.0	0.53	0.00	2.79	2.07
20	7	0.5	18941.0	0.36	0.00	5.81	3.52
20	7	0.6	20645.0	0.00	0.40	0.34	0.09
20	7	0.7	22269.5	0.02	0.00	1.39	5.79
20	7	0.8	23710.0	0.00	0.00	3.32	4.24
20	7	0.9	25082.5	0.00	0.00	2.14	6.17

Table C.9: % Difference: Test Problem No. 9 (n=20)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	4133.8	0.00	0.00	2.91	3.98
20	7	0.2	5999.0	0.98	0.38	0.00	0.15
20	7	0.3	7331.9	0.00	0.06	0.37	0.54
20	7	0.4	8601.2	0.65	0.00	4.19	1.49
20	7	0.5	9856.5	0.91	0.00	3.38	3.05
20	7	0.6	10696.4	0.47	0.00	2.76	4.02
20	7	0.7	11517.3	0.37	0.00	4.21	3.91
20	7	0.8	12336.0	0.23	0.00	4.29	2.12
20	7	0.9	13152.5	0.11	0.00	3.84	2.17

Table C.10: % Difference: Test Problem No. 10 (n=20)

C.1.2 Test Problems with 30 Demand Points

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	12740.0	0.00	0.07	2.59	1.97
30	7	0.4	17632.0	1.62	0.00	5.13	2.44
30	7	0.6	21753.0	0.02	0.00	2.38	1.51
30	7	0.8	25083.4	0.33	0.00	1.84	1.77

Table C.11: % Difference: Test Problem No. 1 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	8991.4	0.00	0.16	3.43	5.19
30	7	0.4	12821.6	0.00	0.00	4.47	3.86
30	7	0.6	15181.6	0.92	0.00	2.72	2.56
30	7	0.8	17189.0	0.00	0.00	3.37	2.06

Table C.12: % Difference: Test Problem No. 2 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	10939.4	0.02	0.00	2.57	2.12
30	7	0.4	15019.4	0.00	0.00	4.64	8.30
30	7	0.6	17948.4	0.00	0.00	1.64	2.19
30	7	0.8	20384.2	0.00	0.00	1.90	3.69

Table C.13: % Difference: Test Problem No. 3 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	8744.2	0.00	0.06	1.87	1.50
30	7	0.4	12450.6	0.00	0.00	3.83	3.22
30	7	0.6	15817.6	1.12	0.00	1.69	4.02
30	7	0.8	18352.8	0.00	0.00	2.76	1.64

Table C.14: % Difference: Test Problem No. 4 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	12759.0	0.00	0.00	2.58	2.97
30	7	0.4	17010.8	1.24	0.00	2.49	1.96
30	7	0.6	19920.6	0.29	0.00	4.12	4.90
30	7	0.8	22453.4	0.00	0.00	2.71	4.67

Table C.15: % Difference: Test Problem No. 5 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	13836.6	1.31	0.00	4.72	7.73
30	7	0.4	20268.6	0.00	0.33	3.05	6.65
30	7	0.6	26491.0	0.98	0.00	0.59	0.27
30	7	0.8	31215.6	0.00	0.00	1.84	1.54

Table C.16: % Difference: Test Problem No. 6 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	9627.0	1.65	0.00	8.21	8.18
30	7	0.4	12528.4	1.22	0.00	10.88	10.01
30	7	0.6	15071.4	0.00	0.00	4.67	4.28
30	7	0.8	16765.2	0.22	0.00	6.12	5.77

Table C.17: % Difference: Test Problem No. 7 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	10092.4	0.00	0.00	4.63	5.30
30	7	0.4	13761.4	0.46	0.00	2.26	5.14
30	7	0.6	16874.4	0.00	0.00	3.13	4.00
30	7	0.8	19371.8	0.70	0.00	0.67	2.11

Table C.18: % Difference: Test Problem No. 8 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	10779.6	0.00	0.00	6.07	11.21
30	7	0.4	14022.0	0.00	0.00	2.94	1.44
30	7	0.6	16100.0	0.00	0.00	5.00	2.59
30	7	0.8	18061.0	0.00	0.00	5.84	6.34

Table C.19: % Difference: Test Problem No. 9 (n=30)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	13540.0	0.60	0.00	1.18	3.60
30	7	0.4	17111.2	0.00	0.00	3.67	6.17
30	7	0.6	19369.6	0.00	0.00	8.83	6.19
30	7	0.8	21543.0	0.00	0.00	4.66	4.66

Table C.20: % Difference: Test Problem No. 10 (n=30)

C.1.3 Test Problems with 40 Demand Points

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	10319.8	1.31	0.00	6.64	2.76
40	5	0.4	13738.8	2.91	0.00	5.90	6.51
40	5	0.6	16590.8	0.00	0.00	6.16	6.13
40	5	0.8	19168.2	0.00	0.00	4.23	3.11

Table C.21: % Difference: Test Problem No. 1 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	15220.2	0.79	0.00	8.77	5.89
40	5	0.4	18943.6	0.00	0.00	8.95	12.16
40	5	0.6	21605.6	0.00	0.01	7.15	8.58
40	5	0.8	23875.4	0.00	0.00	6.65	8.65

Table C.22: % Difference: Test Problem No. 2 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	13124.6	0.00	0.00	5.46	5.68
40	5	0.4	16232.0	0.00	0.30	3.21	3.55
40	5	0.6	18564.6	0.00	0.05	2.12	2.42
40	5	0.8	20532.2	0.00	0.00	2.04	2.36

Table C.23: % Difference: Test Problem No. 3 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	14243.4	0.54	0.54	0.00	3.36
40	5	0.4	18288.8	0.00	0.00	4.79	4.78
40	5	0.6	21228.4	0.00	0.00	3.20	4.96
40	5	0.8	23927.0	0.00	0.00	3.25	2.35

Table C.24: % Difference: Test Problem No. 4 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	15545.6	2.39	0.00	5.44	13.11
40	5	0.4	21222.6	0.00	0.00	1.15	9.37
40	5	0.6	26102.8	0.00	0.00	2.89	5.08
40	5	0.8	30611.4	0.00	0.00	3.67	2.35

Table C.25: % Difference: Test Problem No. 5 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	13150.8	0.00	0.00	4.23	5.24
40	5	0.4	16087.0	0.00	0.00	4.44	8.11
40	5	0.6	18825.0	0.00	0.00	3.00	6.37
40	5	0.8	21553.8	0.00	0.00	4.01	3.91

Table C.26: % Difference: Test Problem No. 6 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	16558.0	0.00	0.63	6.63	5.32
40	5	0.4	23006.6	0.00	0.07	5.96	8.20
40	5	0.6	29153.6	0.04	0.00	2.33	5.37
40	5	0.8	34871.8	0.15	0.00	4.37	4.30

Table C.27: % Difference: Test Problem No. 7 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	13983.4	0.00	0.00	4.82	5.54
40	5	0.4	16394.2	0.00	0.99	7.33	13.53
40	5	0.6	18425.0	0.00	0.00	5.97	5.96
40	5	0.8	20276.0	0.00	0.00	1.99	3.85

Table C.28: % Difference: Test Problem No. 8 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	12287.2	0.15	0.00	11.18	10.20
40	5	0.4	15925.8	0.00	0.00	8.18	5.95
40	5	0.6	18470.4	0.00	0.00	1.70	5.63
40	5	0.8	20515.0	0.00	0.00	4.25	4.10

Table C.29: % Difference: Test Problem No. 9 (n=40)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	11640.6	0.00	0.00	5.06	2.32
40	5	0.4	15963.8	0.00	0.00	5.21	6.05
40	5	0.6	18923.8	0.00	0.44	4.60	8.59
40	5	0.8	22123.4	0.00	0.52	4.23	4.40

Table C.30: % Difference: Test Problem No. 10 (n=40)

C.1.4 Test Problems with 50 Demand Points

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	17089.4	0.73	0.00	4.73	8.32
50	5	0.4	21822.0	0.00	0.00	7.53	11.18
50	5	0.6	25217.2	0.19	0.00	3.25	8.29
50	5	0.8	27768.6	0.00	0.05	2.94	2.30

Table C.31: % Difference: Test Problem No. 1 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	18488.8	0.00	0.00	6.07	7.55
50	5	0.4	22378.8	0.00	0.12	4.58	12.85
50	5	0.6	25694.0	0.09	0.00	9.03	6.22
50	5	0.8	28553.2	0.00	0.00	2.78	2.69

Table C.32: % Difference: Test Problem No. 2 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	12575.0	0.00	0.00	1.95	5.98
50	5	0.4	16044.8	0.00	0.00	11.88	11.80
50	5	0.6	18546.8	0.00	0.00	2.93	4.24
50	5	0.8	20595.6	0.00	0.00	2.69	4.45

Table C.33: % Difference: Test Problem No. 3 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	21484.8	0.01	0.00	7.45	6.64
50	5	0.4	25790.4	0.00	0.77	4.43	4.79
50	5	0.6	29469.6	0.00	0.00	6.08	7.53
50	5	0.8	32919.8	0.00	0.03	4.41	5.55

Table C.34: % Difference: Test Problem No. 4 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	14030.6	0.00	0.66	6.28	10.42
50	5	0.4	17083.0	0.00	0.00	7.63	9.55
50	5	0.6	19691.6	0.00	0.17	8.52	10.82
50	5	0.8	21913.8	0.00	0.00	4.57	5.64

Table C.35: % Difference: Test Problem No. 5 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	18019.2	0.00	0.50	6.90	11.15
50	5	0.4	23668.6	0.00	0.02	3.43	4.14
50	5	0.6	28527.4	0.00	0.44	2.10	2.76
50	5	0.8	31734.2	0.00	0.00	0.59	4.40

Table C.36: % Difference: Test Problem No. 6 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	11461.0	0.00	0.75	7.33	9.25
50	5	0.4	14436.0	0.00	0.30	5.69	6.03
50	5	0.6	17175.4	0.00	0.00	2.75	4.74
50	5	0.8	19611.8	0.13	0.00	3.51	2.69

Table C.37: % Difference: Test Problem No. 7 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	16683.6	0.00	0.35	7.41	9.59
50	5	0.4	20609.0	0.00	0.39	11.76	4.79
50	5	0.6	23679.0	0.00	0.00	5.54	6.49
50	5	0.8	26199.6	0.00	0.05	3.14	5.16

Table C.38: % Difference: Test Problem No. 8 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	16494.8	0.00	1.08	9.56	13.80
50	5	0.4	21073.6	0.00	0.12	1.30	2.79
50	5	0.6	25068.6	0.00	0.00	5.28	3.98
50	5	0.8	28816.2	0.00	0.00	3.90	2.40

Table C.39: % Difference: Test Problem No. 9 (n=50)

n	p	α	Best Sol. Found	% Difference			
				LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	17438.6	0.00	0.00	3.76	6.40
50	5	0.4	23198.0	0.00	0.00	2.18	6.00
50	5	0.6	27908.4	0.00	0.00	1.37	5.39
50	5	0.8	32234.8	0.00	0.00	4.43	4.31

Table C.40: % Difference: Test Problem No. 10 (n=50)

C.2 Comparisons for Computation Times

As discussed earlier, both the LRA type heuristics found solutions much faster than the ARL type heuristics in all the problem instances discussed in this section. ARL(B) seemed to take less time than ARL(A) which may suggest that it gets stuck at a local optimum easier than the ARL(A) heuristic. The computation times for LRA type heuristics between themselves were in general close to one another.

C.2.1 Test Problems with 20 Demand Points

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	12	12	355	294
20	7	0.2	14	13	310	273
20	7	0.3	10	9	320	280
20	7	0.4	8	9	321	253
20	7	0.5	9	8	337	269
20	7	0.6	16	9	310	217
20	7	0.7	9	6	290	225
20	7	0.8	10	8	212	240
20	7	0.9	12	10	205	251

Table C.41: Processing Times: Test Problem No. 1 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	14	11	346	378
20	7	0.2	14	10	380	338
20	7	0.3	12	9	346	291
20	7	0.4	10	8	408	249
20	7	0.5	10	8	310	190
20	7	0.6	9	8	162	231
20	7	0.7	9	7	232	205
20	7	0.8	8	7	195	279
20	7	0.9	6	7	251	259

Table C.42: Processing Times: Test Problem No. 2 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	16	9	230	395
20	7	0.2	18	8	323	307
20	7	0.3	12	6	247	290
20	7	0.4	12	7	258	304
20	7	0.5	9	7	317	268
20	7	0.6	11	6	307	229
20	7	0.7	10	7	338	293
20	7	0.8	5	7	294	251
20	7	0.9	5	7	312	212

Table C.43: Processing Times: Test Problem No. 3 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	14	9	235	352
20	7	0.2	16	10	340	294
20	7	0.3	9	7	267	279
20	7	0.4	7	7	276	308
20	7	0.5	6	7	375	226
20	7	0.6	7	6	312	279
20	7	0.7	5	7	513	254
20	7	0.8	6	7	349	294
20	7	0.9	4	8	434	282

Table C.44: Processing Times: Test Problem No. 4 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	13	12	174	356
20	7	0.2	7	12	213	335
20	7	0.3	8	8	282	283
20	7	0.4	5	7	346	328
20	7	0.5	6	6	330	263
20	7	0.6	4	7	312	292
20	7	0.7	5	7	232	299
20	7	0.8	5	7	313	304
20	7	0.9	6	7	292	319

Table C.45: Processing Times: Test Problem No. 5 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	11	12	205	365
20	7	0.2	12	10	195	331
20	7	0.3	10	7	221	305
20	7	0.4	6	7	285	281
20	7	0.5	7	7	254	297
20	7	0.6	7	8	276	227
20	7	0.7	6	7	347	246
20	7	0.8	5	7	323	271
20	7	0.9	6	7	217	204

Table C.46: Processing Times: Test Problem No. 6 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	15	12	219	336
20	7	0.2	13	7	215	315
20	7	0.3	5	8	231	262
20	7	0.4	5	8	267	223
20	7	0.5	5	6	280	263
20	7	0.6	8	11	329	214
20	7	0.7	7	9	319	312
20	7	0.8	7	10	328	245
20	7	0.9	6	10	323	261

Table C.47: Processing Times: Test Problem No. 7 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	13	11	227	330
20	7	0.2	17	11	276	262
20	7	0.3	17	11	136	252
20	7	0.4	5	7	266	275
20	7	0.5	7	7	219	241
20	7	0.6	11	8	163	217
20	7	0.7	8	7	277	300
20	7	0.8	5	7	268	200
20	7	0.9	5	6	349	244

Table C.48: Processing Times: Test Problem No. 8 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	9	9	218	385
20	7	0.2	9	8	242	263
20	7	0.3	8	9	202	217
20	7	0.4	5	8	268	255
20	7	0.5	6	7	226	232
20	7	0.6	6	6	284	263
20	7	0.7	4	8	250	232
20	7	0.8	5	8	308	257
20	7	0.9	4	8	293	253

Table C.49: Processing Times: Test Problem No. 9 (n=20)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
20	7	0.1	13	12	189	315
20	7	0.2	14	9	425	366
20	7	0.3	15	9	335	279
20	7	0.4	9	8	286	279
20	7	0.5	5	9	280	231
20	7	0.6	6	7	258	240
20	7	0.7	6	7	497	219
20	7	0.8	6	7	413	251
20	7	0.9	5	8	326	250

Table C.50: Processing Times: Test Problem No. 10 (n=20)

C.2.2 Test Problems with 30 Demand Points

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	38	29	865	1394
30	7	0.4	19	20	1884	1195
30	7	0.6	25	28	1082	710
30	7	0.8	19	17	689	859

Table C.51: Processing Times: Test Problem No. 1 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	46	26	1433	1418
30	7	0.4	23	23	1186	960
30	7	0.6	15	17	1127	1067
30	7	0.8	33	27	874	747

Table C.52: Processing Times: Test Problem No. 2 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	40	27	570	1272
30	7	0.4	21	19	1278	937
30	7	0.6	17	17	1556	813
30	7	0.8	15	19	2217	855

Table C.53: Processing Times: Test Problem No. 3 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	49	32	1471	1516
30	7	0.4	50	26	1742	1158
30	7	0.6	15	19	1210	987
30	7	0.8	18	18	1317	1041

Table C.54: Processing Times: Test Problem No. 4 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	40	26	1718	1344
30	7	0.4	19	21	1864	1365
30	7	0.6	23	19	1777	1161
30	7	0.8	20	18	1635	1324

Table C.55: Processing Times: Test Problem No. 5 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	37	25	1822	909
30	7	0.4	39	19	1562	1012
30	7	0.6	23	22	2677	1116
30	7	0.8	15	18	1345	1348

Table C.56: Processing Times: Test Problem No. 6 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	30	20	1326	1188
30	7	0.4	19	25	2127	1271
30	7	0.6	23	19	1578	1084
30	7	0.8	22	21	1565	869

Table C.57: Processing Times: Test Problem No. 7 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	58	29	1951	1002
30	7	0.4	39	25	1750	1023
30	7	0.6	50	27	1513	734
30	7	0.8	14	18	1230	820

Table C.58: Processing Times: Test Problem No. 8 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	60	29	1854	1709
30	7	0.4	40	23	1352	1153
30	7	0.6	29	23	1217	922
30	7	0.8	31	27	1536	795

Table C.59: Processing Times: Test Problem No. 9 (n=30)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
30	7	0.2	39	27	1254	1616
30	7	0.4	11	18	1384	909
30	7	0.6	16	26	772	1042
30	7	0.8	15	26	1087	809

Table C.60: Processing Times: Test Problem No. 10 (n=30)

C.2.3 Test Problems with 40 Demand Points

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	23	33	1347	1042
40	5	0.4	18	29	755	515
40	5	0.6	18	27	424	380
40	5	0.8	18	25	355	289

Table C.61: Processing Times: Test Problem No. 1 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	26	35	679	883
40	5	0.4	26	30	584	545
40	5	0.6	28	39	593	522
40	5	0.8	27	45	651	469

Table C.62: Processing Times: Test Problem No. 2 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	22	35	1662	986
40	5	0.4	25	34	1760	686
40	5	0.6	10	25	1885	1007
40	5	0.8	14	22	1270	863

Table C.63: Processing Times: Test Problem No. 3 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	29	41	1279	815
40	5	0.4	29	37	1152	693
40	5	0.6	21	35	1315	639
40	5	0.8	23	37	586	678

Table C.64: Processing Times: Test Problem No. 4 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	22	37	639	725
40	5	0.4	21	25	890	375
40	5	0.6	13	23	464	413
40	5	0.8	12	24	506	322

Table C.65: Processing Times: Test Problem No. 5 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	26	48	896	780
40	5	0.4	27	45	1213	643
40	5	0.6	23	39	1314	470
40	5	0.8	24	50	651	381

Table C.66: Processing Times: Test Problem No. 6 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	26	41	604	689
40	5	0.4	28	37	489	512
40	5	0.6	22	37	825	398
40	5	0.8	29	56	251	311

Table C.67: Processing Times: Test Problem No. 7 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	27	38	1795	621
40	5	0.4	25	25	962	422
40	5	0.6	21	25	864	581
40	5	0.8	20	23	1642	757

Table C.68: Processing Times: Test Problem No. 8 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	30	39	623	709
40	5	0.4	21	33	584	454
40	5	0.6	14	33	1155	467
40	5	0.8	13	32	811	420

Table C.69: Processing Times: Test Problem No. 9 (n=40)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
40	5	0.2	26	36	772	936
40	5	0.4	18	23	694	566
40	5	0.6	23	28	1499	503
40	5	0.8	17	22	1113	539

Table C.70: Processing Times: Test Problem No. 10 (n=40)

C.2.4 Test Problems with 50 Demand Points

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	42	24	3373	1530
50	5	0.4	43	31	1737	944
50	5	0.6	36	19	488	761
50	5	0.8	45	16	2201	1337

Table C.71: Processing Times: Test Problem No. 1 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	62	24	445	2383
50	5	0.4	60	17	3096	1047
50	5	0.6	47	16	1839	1065
50	5	0.8	40	17	2240	1247

Table C.72: Processing Times: Test Problem No. 2 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	52	31	2797	1634
50	5	0.4	63	23	1128	768
50	5	0.6	50	24	2388	1006
50	5	0.8	49	24	1284	741

Table C.73: Processing Times: Test Problem No. 3 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	39	32	415	1992
50	5	0.4	46	28	3277	1443
50	5	0.6	40	23	2167	979
50	5	0.8	41	17	2566	1176

Table C.74: Processing Times: Test Problem No. 4 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	58	24	3202	1862
50	5	0.4	58	21	4046	1649
50	5	0.6	53	23	821	950
50	5	0.8	28	16	1162	875

Table C.75: Processing Times: Test Problem No. 5 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	62	30	2043	2031
50	5	0.4	53	22	35	2775
50	5	0.6	36	24	3590	1554
50	5	0.8	28	24	3890	1313

Table C.76: Processing Times: Test Problem No. 6 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	42	21	1817	1109
50	5	0.4	43	24	1594	962
50	5	0.6	29	24	2698	1093
50	5	0.8	25	29	1489	933

Table C.77: Processing Times: Test Problem No. 7 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	34	24	3315	2139
50	5	0.4	44	24	1463	1792
50	5	0.6	35	24	963	878
50	5	0.8	33	27	2928	683

Table C.78: Processing Times: Test Problem No. 8 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	49	27	3032	1498
50	5	0.4	45	25	3422	1397
50	5	0.6	45	21	3586	1280
50	5	0.8	43	16	1078	1108

Table C.79: Processing Times: Test Problem No. 9 (n=50)

n	p	α	Average CPU (in seconds)			
			LRA(A)	LRA(B)	ARL(A)	ARL(B)
50	5	0.2	43	25	3981	2838
50	5	0.4	40	17	2572	1512
50	5	0.6	27	16	3071	1188
50	5	0.8	28	16	2102	1104

Table C.80: Processing Times: Test Problem No. 10 (n=50)

Appendix D

Comparisons for Each Test Problem: LRA(A) vs LRA(B)

In this appendix, we compare the two LRA type heuristics for even larger problem instances. For smaller instances, in the previous appendices, LRA(A) seemed to find better results than LRA(B) more often. However, the results in this appendix show that LRA(B) performs better when the problem size increases. Especially when the value of p increases with respect to n, LRA(B) begins to perform more favorably.

D.1 Test Problems with 30 Demand Points

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	12587.4	0.00	0.00	160	102
30	10	0.4	17620.8	4.16	0.00	46	69
30	10	0.6	21757.8	0.19	0.00	52	74
30	10	0.8	24979.0	0.75	0.00	45	63

Table D.1: Test Problem No. 1 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	8765.8	1.26	0.00	115	75
30	10	0.4	12612.2	1.66	0.00	55	73
30	10	0.6	15181.6	0.92	0.00	45	67
30	10	0.8	17189.0	0.00	0.00	88	97

Table D.2: Test Problem No. 2 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	10879.8	0.27	0.00	98	81
30	10	0.4	14971.4	0.32	0.00	53	69
30	10	0.6	17948.4	0.00	0.00	42	62
30	10	0.8	20384.2	0.00	0.00	42	62

Table D.3: Test Problem No. 3 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	8529.8	0.00	0.00	217	104
30	10	0.4	12345.6	0.00	0.00	136	94
30	10	0.6	15800.0	0.00	0.00	52	64
30	10	0.8	18349.2	0.02	0.00	48	63

Table D.4: Test Problem No. 4 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	12502.2	0.00	1.57	121	79
30	10	0.4	17010.8	1.24	0.00	47	69
30	10	0.6	19920.6	0.29	0.00	64	63
30	10	0.8	22453.4	0.00	0.00	60	64

Table D.5: Test Problem No. 5 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	13965.6	0.05	0.00	150	87
30	10	0.4	20216.6	0.00	0.58	87	71
30	10	0.6	26751.4	0.11	0.00	53	72
30	10	0.8	31215.6	0.00	0.00	39	63

Table D.6: Test Problem No. 6 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	9785.8	0.00	0.00	57	72
30	10	0.4	12528.4	1.22	0.00	51	82
30	10	0.6	15071.4	0.00	0.00	51	76
30	10	0.8	16765.2	0.22	0.00	55	79

Table D.7: Test Problem No. 7 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	9711.2	0.00	2.05	122	83
30	10	0.4	13677.0	0.07	0.00	86	75
30	10	0.6	16868.8	0.00	0.00	118	82
30	10	0.8	19422.6	0.00	0.00	42	62

Table D.8: Test Problem No. 8 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	10665.6	0.00	0.00	163	82
30	10	0.4	14022.0	0.00	0.00	93	74
30	10	0.6	16100.0	0.00	0.00	83	96
30	10	0.8	18061.0	0.00	0.00	78	83

Table D.9: Test Problem No. 9 (n=30)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
30	10	0.2	13602.4	0.14	0.00	96	71
30	10	0.4	17111.2	0.00	0.00	37	63
30	10	0.6	19369.6	0.00	0.00	39	90
30	10	0.8	21543.0	0.00	0.00	44	89

Table D.10: Test Problem No. 10 (n=30)

D.2 Test Problems with 40 Demand Points

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	9853.8	0.00	0.1	80	132
40	7	0.4	13572.0	0.00	0.14	95	143
40	7	0.6	16658.0	0.00	0.00	45	78
40	7	0.8	19046.6	0.64	0.00	45	78
40	10	0.2	9482.6	0.68	0.00	314	443
40	10	0.4	13520.4	0.00	0.01	317	549
40	10	0.6	16658	0.00	0.00	94	289
40	10	0.8	19046.6	0.64	0.00	110	313
40	12	0.2	9482.6	1.76	0.00	504	840
40	12	0.4	13520.4	0.00	0.00	495	942
40	12	0.6	16648.0	0.06	0.00	201	652
40	12	0.8	19046.6	0.64	0.00	206	611

Table D.11: Test Problem No. 1 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	14422.2	0.65	0.00	126	128
40	7	0.4	18661.6	0.00	0.00	103	99
40	7	0.6	21524.0	0.00	0.00	69	89
40	7	0.8	23834.6	0.00	0.00	71	81
40	10	0.2	13755.8	1.57	0.00	488	483
40	10	0.4	18654.4	0.00	0.00	317	390
40	10	0.6	21524.0	0.00	0.00	202	460
40	10	0.8	23834.6	0.00	0.00	177	397
40	12	0.2	13606.6	0.37	0.00	658	733
40	12	0.4	18654.4	0.00	0.00	488	720
40	12	0.6	21524.0	0.00	0.00	304	718
40	12	0.8	23834.6	0.00	0.00	296	577

Table D.12: Test Problem No. 2 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	12715.0	1.14	0.00	84	118
40	7	0.4	16179.8	0.00	0.00	80	93
40	7	0.6	18564.6	0.00	0.00	32	77
40	7	0.8	20532.2	0.00	0.00	35	79
40	10	0.2	12453.0	3.67	0.00	200	317
40	10	0.4	16145.0	0.13	0.00	233	315
40	10	0.6	18564.6	0.00	0.05	86	283
40	10	0.8	20532.2	0.00	0.00	85	280
40	12	0.2	12453.0	2.82	0.00	286	569
40	12	0.4	16136.8	0.18	0.00	345	609
40	12	0.6	18564.6	0.00	0.00	157	544
40	12	0.8	20532.2	0.00	0.00	135	546

Table D.13: Test Problem No. 3 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	13491.2	0.00	0.00	109	129
40	7	0.4	17963.6	0.00	0.46	123	117
40	7	0.6	21153.2	0.00	0.00	51	81
40	7	0.8	23889.4	0.00	0.00	55	76
40	10	0.2	13271.6	0.00	0.00	307	336
40	10	0.4	17954.0	0.05	0.00	210	306
40	10	0.6	21153.2	0.00	0.00	136	333
40	10	0.8	23889.4	0.00	0.00	138	285
40	12	0.2	13271.6	0.00	0.00	513	673
40	12	0.4	17954.0	0.05	0.00	292	556
40	12	0.6	21153.2	0.00	0.00	226	555
40	12	0.8	23889.4	0.00	0.00	258	555

Table D.14: Test Problem No. 4 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	14661.2	0.62	0.00	98	155
40	7	0.4	21052.4	0.00	0.00	101	133
40	7	0.6	26102.8	0.00	0.00	36	103
40	7	0.8	30611.4	0.00	0.00	30	77
40	10	0.2	14366.8	0.00	0.22	441	321
40	10	0.4	20957.4	0.00	0.00	214	444
40	10	0.6	26102.8	0.00	0.00	100	381
40	10	0.8	30611.4	0.00	0.00	88	299
40	12	0.2	14350.6	0.00	0.00	725	763
40	12	0.4	20957.4	0.00	0.00	341	885
40	12	0.6	26102.8	0.00	0.00	158	684
40	12	0.8	30611.4	0.00	0.00	157	545

Table D.15: Test Problem No. 5 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	12608.2	0.00	0.00	109	142
40	7	0.4	16032.6	0.00	0.02	71	118
40	7	0.6	18791.4	0.00	0.00	61	117
40	7	0.8	21537.0	0.00	0.00	60	112
40	10	0.2	12271.8	0.00	0.00	411	729
40	10	0.4	16032.6	0.00	0.00	189	394
40	10	0.6	18791.4	0.00	0.00	143	419
40	10	0.8	21537.0	0.00	0.00	163	428
40	12	0.2	12271.8	0.00	0.00	591	1069
40	12	0.4	15916.2	0.73	0.00	290	698
40	12	0.6	18791.4	0.00	0.00	243	781
40	12	0.8	21498.2	0.18	0.00	248	807

Table D.16: Test Problem No. 6 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	15835.8	0.00	0.87	120	146
40	7	0.4	22775.0	0.00	0.08	100	135
40	7	0.6	28840.6	0.71	0.00	76	151
40	7	0.8	34709.4	0.00	0.00	109	141
40	10	0.2	15329.8	0.00	0.00	459	428
40	10	0.4	22770.4	0.1	0.00	253	409
40	10	0.6	28977.0	0.24	0.00	199	524
40	10	0.8	34670.2	0.00	0.00	369	686
40	12	0.2	15271.4	0.38	0.00	703	597
40	12	0.4	22770.4	0.1	0.00	400	819
40	12	0.6	28840.6	0.00	0.57	270	835
40	12	0.8	34670.2	0.00	0.00	588	1426

Table D.17: Test Problem No. 7 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	13099.0	0.00	0.23	113	142
40	7	0.4	16177.0	1.34	0.00	53	93
40	7	0.6	18425.0	0.00	0.00	46	81
40	7	0.8	20276.0	0.00	0.00	47	80
40	10	0.2	12521.6	0.00	0.07	579	459
40	10	0.4	16177.0	1.34	0.00	159	297
40	10	0.6	18425.0	0.00	0.00	132	281
40	10	0.8	20276.0	0.00	0.00	139	281
40	12	0.2	12411.2	0.00	0.00	793	676
40	12	0.4	16394.2	0.00	0.00	257	606
40	12	0.6	18425.0	0.00	0.00	212	611
40	12	0.8	20276.0	0.00	0.00	276	552

Table D.18: Test Problem No. 8 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	11390.8	0.00	1.35	131	138
40	7	0.4	15652.4	0.00	0.11	72	107
40	7	0.6	18470.4	0.00	0.00	34	114
40	7	0.8	20515.0	0.00	0.00	34	118
40	10	0.2	11005.2	2.47	0.00	309	466
40	10	0.4	15607.0	0.29	0.00	169	315
40	10	0.6	18466.8	0.02	0.00	90	376
40	10	0.8	20515.0	0.00	0.00	96	367
40	12	0.2	11005.2	5.56	0.00	512	982
40	12	0.4	15607.0	0.29	0.00	306	787
40	12	0.6	18466.8	0.02	0.00	140	725
40	12	0.8	20515.0	0.00	0.00	160	784

Table D.19: Test Problem No. 9 (n=40)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
40	7	0.2	11339.4	0.00	0.00	99	154
40	7	0.4	15317.6	4.13	0.00	62	82
40	7	0.6	18833.4	0.00	0.00	57	79
40	7	0.8	22102.2	0.00	0.00	52	93
40	10	0.2	10765.2	3.21	0.00	588	416
40	10	0.4	15307.4	4.2	0.00	168	289
40	10	0.6	18833.4	0.00	0.00	141	288
40	10	0.8	22102.2	0.00	0.00	146	315
40	12	0.2	11106.8	0.00	0.00	1110	757
40	12	0.4	15307.4	4.2	0.00	282	557
40	12	0.6	18833.4	0.00	0.00	227	667
40	12	0.8	22102.2	0.00	0.00	208	609

Table D.20: Test Problem No. 10 (n=40)

D.3 Test Problems with 50 Demand Points

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	16451.0	0.00	0.00	136	103
50	7	0.4	21408.6	0.00	0.00	190	107
50	7	0.6	25086.2	0.00	0.00	129	61
50	7	0.8	27768.6	0.00	0.00	107	55
50	10	0.2	15861.2	0.00	0.52	990	425
50	10	0.4	21275.8	0.39	0.00	504	263
50	10	0.6	25086.2	0.00	0.00	301	263
50	10	0.8	27768.6	0.00	0.00	290	211
50	12	0.2	15839.8	0.00	0.05	1680	581
50	12	0.4	21313.8	0.00	0.21	827	518
50	12	0.6	25086.2	0.00	0.31	478	433
50	12	0.8	27768.6	0.00	0.00	455	429
50	15	0.2	15618.0	1.42	0.00	3065	1451
50	15	0.4	21323.0	0.17	0.00	1423	1250
50	15	0.6	25086.2	0.00	0.31	823	1016
50	15	0.8	27768.6	0.00	0.00	866	921

Table D.21: Test Problem No. 1 (n=50)

n	p	α	Best Sol. Found	% Difference LRA(A)	% Difference LRA(B)	CPU (in seconds) LRA(A)	CPU (in seconds) LRA(B)
50	7	0.2	17757.2	0.00	0.59	248	97
50	7	0.4	22045.4	0.00	0.13	203	115
50	7	0.6	25636.6	0.00	0.00	124	56
50	7	0.8	28553.2	0.00	0.00	102	56
50	10	0.2	17491.4	0.00	1.05	847	307
50	10	0.4	21986.8	0.00	0.00	579	309
50	10	0.6	25426.0	0.00	0.83	354	227
50	10	0.8	28553.2	0.00	0.00	275	214
50	12	0.2	17491.4	0.00	0.00	1185	498
50	12	0.4	21986.8	0.00	0.00	880	571
50	12	0.6	25426.0	0.83	0.00	478	433
50	12	0.8	28553.2	0.00	0.00	450	413
50	15	0.2	17366.6	0.00	1.69	2164	989
50	15	0.4	21902.2	0.39	0.00	1704	1171
50	15	0.6	25426.0	0.00	0.00	1031	1073
50	15	0.8	28553.2	0.00	0.00	798	928

Table D.22: Test Problem No. 2 (n=50)

n	p	α	Best Sol. Found	% Difference LRA(A)	% Difference LRA(B)	CPU (in seconds) LRA(A)	CPU (in seconds) LRA(B)
50	7	0.2	11928.6	0.00	0.00	262	97
50	7	0.4	15716.0	0.00	0.00	250	81
50	7	0.6	18463.6	0.00	0.00	133	78
50	7	0.8	20554.0	0.00	0.00	134	56
50	10	0.2	11617.4	0.78	0.00	946	330
50	10	0.4	15580.8	1.05	0.00	611	213
50	10	0.6	18463.6	0.00	0.00	396	263
50	10	0.8	20554.0	0.00	0.00	335	212
50	12	0.2	11708.4	0.00	0.00	1281	552
50	12	0.4	15580.8	0.00	1.05	1132	421
50	12	0.6	18463.6	0.00	0.00	653	555
50	12	0.8	20554.0	0.00	0.00	620	414
50	15	0.2	11701.0	0.06	0.00	2261	1167
50	15	0.4	15580.8	0.00	0.45	1818	1076
50	15	0.6	18463.6	0.00	0.00	1199	1254
50	15	0.8	20554.0	0.00	0.00	1028	936

Table D.23: Test Problem No. 3 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	20436.8	0.00	0.49	200	91
50	7	0.4	25548.0	0.00	0.06	145	75
50	7	0.6	29252.0	0.00	0.02	152	81
50	7	0.8	32794.6	0.26	0.00	123	76
50	10	0.2	19680.4	0.00	0.34	804	339
50	10	0.4	25470.4	0.3	0.00	405	253
50	10	0.6	29252.0	0.00	0.74	410	249
50	10	0.8	32792.0	0.00	0.00	350	216
50	12	0.2	19680.4	0.00	1.29	1336	623
50	12	0.4	25470.4	0.3	0.00	693	513
50	12	0.6	29252.0	0.00	0.00	601	429
50	12	0.8	32792.0	0.00	0.00	616	407
50	15	0.2	19526.8	0.79	0.00	2008	1315
50	15	0.4	25470.4	0.3	0.00	1301	1352
50	15	0.6	29252.0	0.00	0.00	1203	1061
50	15	0.8	32792.0	0.00	0.00	1080	962

Table D.24: Test Problem No. 4 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	13216.4	1.28	0.00	223	109
50	7	0.4	16668.0	0.00	0.00	192	91
50	7	0.6	19549.6	0.33	0.00	145	56
50	7	0.8	21930.4	0.00	0.00	75	56
50	10	0.2	12921.2	1.08	0.00	733	371
50	10	0.4	16628.4	0.47	0.00	439	268
50	10	0.6	19530.4	0.1	0.00	367	222
50	10	0.8	21858.6	0.00	0.00	220	207
50	12	0.2	12810.8	1.91	0.00	1225	588
50	12	0.4	16628.4	0.00	0.00	842	475
50	12	0.6	19530.4	0.00	0.00	619	452
50	12	0.8	21826.6	0.15	0.00	422	412
50	15	0.2	12912.4	1.11	0.00	2004	1121
50	15	0.4	16587.0	0.72	0.00	1503	1163
50	15	0.6	19469.2	0.41	0.00	1078	981
50	15	0.8	21826.6	0.15	0.00	841	930

Table D.25: Test Problem No. 5 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	17148.4	0.00	0.31	268	91
50	7	0.4	23403.8	0.00	0.00	197	73
50	7	0.6	28432.4	0.00	0.18	131	72
50	7	0.8	31712.6	0.00	0.00	79	55
50	10	0.2	16833.2	1.25	0.00	676	263
50	10	0.4	23306.0	0.00	0.00	581	237
50	10	0.6	28407.2	0.00	0.09	442	304
50	10	0.8	31712.6	0.00	0.00	209	209
50	12	0.2	16833.2	1.25	0.00	1090	520
50	12	0.4	23306.0	0.00	0.00	953	500
50	12	0.6	28407.2	0.00	0.09	705	600
50	12	0.8	31712.6	0.00	0.00	342	412
50	15	0.2	16833.2	1.25	0.00	1795	987
50	15	0.4	23306.0	0.00	0.00	1915	1078
50	15	0.6	28407.2	0.00	0.09	1210	1114
50	15	0.8	31712.6	0.00	0.00	663	924

Table D.26: Test Problem No. 6 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	11041.8	0.00	0.00	223	96
50	7	0.4	14232.6	0.00	0.35	140	111
50	7	0.6	17006.0	1	0.00	81	94
50	7	0.8	19604.6	0.00	0.05	69	72
50	10	0.2	10794.2	0.5	0.00	679	305
50	10	0.4	14092.8	0.61	0.00	485	291
50	10	0.6	17026.4	0.00	0.01	202	249
50	10	0.8	19598.2	0.00	0.00	196	364
50	12	0.2	10772.4	0.71	0.00	1206	579
50	12	0.4	14084.0	0.67	0.00	681	677
50	12	0.6	17028.4	0.86	0.00	319	470
50	12	0.8	19598.2	0.2	0.00	310	614
50	15	0.2	10722.2	1.38	0.00	2122	1214
50	15	0.4	14084.0	0.67	0.00	1127	1586
50	15	0.6	17026.4	0.00	0.01	803	972
50	15	0.8	19598.2	0.00	0.00	642	1118

Table D.27: Test Problem No. 7 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	15892.0	0.00	0.29	184	92
50	7	0.4	20100.2	0.00	0.29	151	104
50	7	0.6	23409.0	0.04	0.00	146	125
50	7	0.8	26111.8	0.00	0.00	134	125
50	10	0.2	15426.0	0.48	0.00	684	275
50	10	0.4	19956.2	0.12	0.00	336	304
50	10	0.6	23330.2	0.00	0.00	480	426
50	10	0.8	26103.4	0.00	0.00	377	396
50	12	0.2	15287.6	0.91	0.00	1067	411
50	12	0.4	19956.2	0.12	0.00	672	535
50	12	0.6	23330.2	0.00	0.00	860	728
50	12	0.8	26103.4	0.00	0.00	624	725
50	15	0.2	15287.6	0.91	0.00	2065	932
50	15	0.4	19956.2	0.12	0.00	1541	1494
50	15	0.6	23330.2	0.00	0.00	1469	1825
50	15	0.8	26103.4	0.00	0.00	1158	1592

Table D.28: Test Problem No. 8 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	15802.0	0.00	0.18	236	92
50	7	0.4	20865.2	0.28	0.00	154	83
50	7	0.6	25068.6	0.00	0.00	109	65
50	7	0.8	28816.2	0.00	0.00	101	61
50	10	0.2	15423.6	1.45	0.00	572	316
50	10	0.4	20820.8	0.49	0.00	377	310
50	10	0.6	25068.6	0.00	0.00	326	252
50	10	0.8	28816.2	0.00	0.00	259	213
50	12	0.2	15275.8	2.43	0.00	860	642
50	12	0.4	20891.6	0.00	0.15	599	544
50	12	0.6	25068.6	0.00	0.00	487	455
50	12	0.8	28816.2	0.00	0.00	446	414
50	15	0.2	15287.6	0.89	0.00	1440	1264
50	15	0.4	20820.8	0.34	0.00	1042	1172
50	15	0.6	25068.6	0.00	0.00	891	979
50	15	0.8	28816.2	0.00	0.00	871	934

Table D.29: Test Problem No. 9 (n=50)

n	p	α	Best Sol. Found	% Difference		CPU (in seconds)	
				LRA(A)	LRA(B)	LRA(A)	LRA(B)
50	7	0.2	16754.6	0.00	0.00	193	79
50	7	0.4	22870.4	0.44	0.00	88	65
50	7	0.6	27861.0	0.00	0.17	63	55
50	7	0.8	32234.8	0.00	0.00	61	54
50	10	0.2	16376.2	0.84	0.00	697	265
50	10	0.4	22870.4	0.00	0.00	263	221
50	10	0.6	27846.0	0.22	0.00	182	209
50	10	0.8	32234.8	0.00	0.00	178	212
50	12	0.2	16376.2	0.84	0.00	987	461
50	12	0.4	22870.4	0.00	0.00	463	437
50	12	0.6	27908.4	0.00	0.00	324	411
50	12	0.8	32234.8	0.00	0.00	313	411
50	15	0.2	16376.2	2.11	0.00	1643	1077
50	15	0.4	22870.4	0.00	0.00	922	932
50	15	0.6	27846.0	0.00	0.00	654	922
50	15	0.8	32234.8	0.00	0.00	579	929

Table D.30: Test Problem No. 10 (n=50)

Bibliography

Aarst, E. H. L. and J. Korst (1989a). *Simulated Annealing and Boltzman Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* Chichester: John Wiley & Sons.

Aarst, E. H. L. and J. Korst (1989b). Boltzman machines for traveling salesman problems. *European Journal of Operational Research 39*, 79–95.

Afrati, F., S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou (1986). The complexity of the traveling repairman problem. *Informatique Th'eorique et Applications 20*, 79–87.

Arthur, J. L. and J. O. Frendeway (1985). A computational study of tour construction procedures for the traveling salesman problem. Research Report, Oregon State University, Corvallis.

Balas, E. and M. W. Padberg (1976). Set partitioning: A survey. *SIAM Review 18*, 710–760.

Balas, E. and P. Toth (1985). Branch and bound methods. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem*, pp. 361–401. Chichester: John Wiley & Sons.

Balinski, M. L. (1965). Integer programming: Methods, uses, computation.

Management Science 12, 253–313.

Balinski, M. L. and P. Wolfe (1963). On benders decomposition and a plant location problem. Technical report, Working Paper ARO-27. Mathematica Inc.,Princeton.

Bellmore, M. and J. C. Malone (1971). Pathology of traveling salesman subtour-elimination algorithms. *Operations Research 19*, 278–307.

Bentley, J. L. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing 4*, 387–411.

Bianco, L., A. Mingozzi, and S. Ricciardelli (1993). The traveling salesman problem with cumulative costs. *Networks 11*, 145–164.

Biggs, N. L., E. K. Lloyd, and R. J. Wilson (1976). *Graph Theory*. Oxford: Clarendon Press.

Bilde, O. and J. Krarup (1977). Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Mathematics 1*, 79–97.

Blum, A., C. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan (1994). The minimum latency problem. In *Proc. 26th ACM Symposium on the Theory of Computing*, pp. 163–171.

Boyd, S. C. and W. H. Cunningham (1991). Small traveling salesman polytopes. *Mathematics of Operations Research 16*, 259–271.

Boyd, S. C. and W. R. Pulleyblank (1990). Optimizing over the subtour polytope of the traveling salesman problem. *Mathematical Programming 49*, 163–188.

Brandeau, M. S. and S. S. Chiu (1989). Overview of representative problems in location research. *Management Science 35*, 645–674.

Carpaneto, G. and P. Toth (1980). Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science 26*, 736–743.

Cerny, V. (1985). A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications 45*, 233–254.

Chhajed, D., R. Francis, and T. Lowe (1993). Contributions of operations research to location science. *Location Science 1*, 263–287.

Christof, T., M. Jünger, and G. Reinelt (1991). A complete description of the traveling salesman polytope on 8 nodes. *Operations Research Letters 10*, 497–500.

Christofides, N. (1970). The shortest Hamiltonian chain of a graph. *SIAM Journal on Applied Mathematics 19*, 689–696.

Christofides, N. (1976). Worst case analysis of a new heuristic for the traveling salesman problem. Research Report, Carnegie Mellon University, Pittsburgh.

Christofides, N., A. Mingozzi, and P. Toth (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks 11*, 145–164.

Church, R. L, J. R. Current, and H. A. Eiselt (1993). Editorial *Location Science 1*, 1–3.

Clarke, G. and J. W. Wright (1964). Scheduling of vehicles from a central depot to

a number of delivery points. *Operations Research 12*, 568–581.

Collins, N. E., R. W. Eglese, and B. L. Golden (1988). Simulated annealing: An annotated bibliography. *American Journal of Math. and Management Science 8*, 205–307.

Cornuejols, G., M. L. Fisher, and G. L. Nemhauser (1977). Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science 23*, 789–810.

Cornuéjols, G. and G. L. Nemhauser (1978). Tight bounds for Christofides' traveling salesman heuristic. *Mathematical Programming 14*, 116–121.

Cornuejols, G., G. L. Nemhauser, and L. A. Wolsey (1990). The uncapacitated facility location problem. In P. B. Mirchandani and R. L. Francis (Eds.), *Discrete Location Theory*, pp. 119–171. New York: Wiley Interscience.

Cornuejols, G. and J. M. Thizy (1982). A primal approach to the simple plant location problem. *SIAM Journal on Algebraic and Discrete Methods 3*, 504–510.

Croes, G. A. (1958). A method for solving traveling salesman problems. *Operations Research 6*, 791–812.

Crowder, H. and M. W. Padberg (1980). Solving large-scale symmetric traveling salesman problems to optimality. *Management Science 26*, 495–509.

Damberg, O. and A. Migdalas (1994). Simulated annealing, local search and Lagrangean heuristics for concentrator location in centralized communication networks: a comparative study. Technical report, LiTH-MAT-R-1994-21.

Linköpings Universitet, Linköping, Sweden.

Dantzig, G. B., D. R. Fulkerson, and S. M. Johnson (1954). Solution of a large scale traveling-salesman problem. *Operations Research 2*, 393–410.

Daskin, M. (1995). *Network and Discrete Location: Models, Algorithms and Applications*. Wiley-Interscience, New York.

Dearing, P. M., R. L. Francis, and T. J. Lowe (1976). Convex location problems on tree networks. *Operations Research 24*, 628–641.

Domschke, W. and A. Drexl (1985). *Location and Layout Planning* Sipringer-Verlag, Berlin.

Drezner, Z. (Ed.) (1995). *Facility Location: A Survey of Applications and Methods*. Springer-Verlag.

Durbin, R. and D. Willshaw (1987). An analogue approach to the traveling salesman problem using an elastic net method. *Nature 326*, 689–691.

Efroymson, M. A. and T. L. Ray (1966). A branch and bound algorithm for plant location. *Operations Research 14*, 361–368.

Erlenkotter, D. (1978). A dual based procedure for uncapacitated facility location. *Operations Research 26*, 992–1009.

Euler, L. (1759). Solution d'une question curieuse qui ne paroit soumise à aucune analyse. *Mem. Acad. Sci. Berlin 15*, 310–337.

Fischetti, M., G. Laporte, and S. Martello (1993). The delivery man problem and cumulative matroids. *Operations Research 41*, 1055–1076.

Fischetti, M. and P. Toth (1993). An efficient algorithm for the min-sum arborescence problem on complete directed graphs. *Mathematical Programming 53*, 173–197.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research 4*, 61–75.

Fox, K. R., B. Gavish, and S. C. Graves (1980). An n-constraint formulation of the (time-dependent) traveling salesman problem. *Operations Research 28*, 1018–1021.

Francis, R. L. and J. M. Goldstein (1974). Location theory: A selective bibliography. *Operations Research 22*, 400–410.

Francis, R. L., J. L. F. McGinnis, and J. A. White (1992). *Facility Layout and Location: An Analytical Approach*. Prentice-Hall, Englewood Cliffs, New Jersey.

Frieze, A. M. (1979). Worst-case analysis of algorithms for traveling salesman problems. *OR Verfahren 32*, 93–112.

Fritzke, B. and P. Wilke (1991). FLEXMAP - a neural network for the traveling salesman problem with linear time and space complexity. In *Proc. International Joint Conference on Neural Networks*, Singapore, pp. 929–934.

Galvão, R. D. (1980). A dual-bounded algorithm for the p-median problem. *Operations Research 28*, 1112–1121.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. San Francisco, California: Freeman.

Garfinkel, R. S. (1985). Motivation and modeling. In E. L. Lawler, J. K. Lenstra,

A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem.* Chichester: John Wiley & Sons.

Garfinkel, R. S., A. W. Neebe, and M. R. Rao (1974). An algorithm for the m-median plant location problem. *Transportation Science 8*, 217–236.

Gavish, B. and K. Srikanth (1986). An optimal method for large-scale multiple traveling salesman problem. *Operations Research 34*, 698–717.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study 2*, 82–114.

Glover, F. (1989). Tabu search. *ORSA Journal on Computing 2*, 4–32 (Part II).

Goemans, M. and J. Kleinberg (1996). An improved approximation ratio for the minimum latency problem. In *Proc. 7th Annual ACM-SIAM Symp. on Disc. Algorithms*, pp. 152–158.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

Golden, B. L., L. D. Bodin, T. Doyle, and W. Stewart, Jr. (1980). Approximate traveling salesman algorithms. *Operations Research 28*, 694–711.

Golden, B. L. and W. R. Stewart (1985). Emprical analysis of heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem.* Chichester: John Wiley & Sons.

Goldman, A. J. (1971). Optimal center location in simple networks. *Transportation Science 5*, 212–221.

Gouveia, L. and S. Voß (1995). A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research 83*, 69–82.

Grötschel, M. (1980). On the symmetric traveling salesman problem: solution of a 120-city problem. *Mathematical Programming Studies 12*, 61–77.

Grötschel, M. and O. Holland (1991). Solution of large-scale symmetric traveling salesman problems. *Mathematical Programming 51*, 141–202.

Grötschel, M., L. Lovász, and A. Schrijver (1988). *Geometric Algorithms and Combinatorial Optimization*. Heidelberg: Springer.

Grötschel, M. and M. W. Padberg (1985). Polyhedral theory. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem*, pp. 361–401. Chichester: John Wiley & Sons.

Guignard, M. and K. Spielberg (1977). Algorithms for exploiting the structure of the simple plant location problem. *Annals of Discrete Mathematics 1*, 247–271.

Hadley, G. (1964). *Nonlinear and Dynamic Programming*. Reading, MA: Addison-Wesley.

Hakimi, S. L. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research 13*, 462–475.

Handler, G. Y. and P. B. Mirchandani (1979). *Location in Networks: Theory and Algorithms*. Cambridge, Massachusetts: M.I.T. Press.

Hansen, P., E. L. Pedrosa Filho, and C. C. Ribeiro (1992). Location and sizing

of offshore platforms for oil exploration. *European Journal of Operational Research 58*, 202–214.

Helbig-Hansen, K. H. and J. Krarup (1974). Improvements of the held-karp algorithm for the symmetric traveling salesman problem. *Mathematical Programming 7*, 87–96.

Held, M. and R. M. Karp (1970). The traveling salesman problem and minimum spanning trees. *Operations Research 18*, 1138–1162.

Held, M. and R. M. Karp (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1*, 6–25.

Hoffman, A. J. and P. Wolfe (1985). History. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem*. Chichester: John Wiley & Sons.

Houck, D. J., J. C. Picard, M. Queyranne, and R. R. Vemuganti (1980). The traveling salesman problem as a constrained shortest path problem: Theory and computational experience. *OPSEARCH 17*, 94–109.

Hurter, A. P. J. and J. S. Martinich (1989). *Facility Location and Theory of Production*. Boston, Dordrect, London: Kluwer Academic Publishers.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In G. Goos and J. Hartmanis (Eds.), *Automata, Languages and Programming*, pp. 446–461. Heidelberg: Lecture Notes in Computer Science 442, Springer.

Johnson, D. S., C. R. Aragon, L. A. Mcgeoch, and C. Schevon (1991). Optimization by simulated annealing: An experimental evaluation. *Operations Research 37*,

865–892 (Part I). *Operations Research*, 39:378–406 (Part II).

Johnson, D. S. and C. H. Papadimitriou (1985). Computational complexity. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem*. Chichester: John Wiley & Sons.

Jünger, M., G. Reinelt, and G. Rinaldi (1994). The traveling salesman problem. In M. Ball, T. Magnanti, C. L. Monma, and G. Nemhauser (Eds.), *Handbook on Operations Research and Management Sciences: Networks*, pp. 446–461. North Holland.

Kariv, O. and S. L. Hakimi (1979). An algorithmic approach to network location problems, part II: the p-medians. *SIAM Journal of Applied Mathematics 37*, 539–560.

Khumawala, B. M. (1972). An efficient branch and bound algorithm for the warehouse location problem. *Management Science 18*, B718–B731.

Kirkman, T. P. (1856). On the representation of polyhedra. *Philos. Trans. Roy. Soc. London Ser. A 146*, 413–418.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics 34*, 975–986.

Knox, J. and F. Glover (1989). Comparative testing of traveling salesman heuristics derived from tabu search, genetic algorithms and simulated annealing. Technical report, University of Colorado.

Kolen, A. (1983). Solving covering problems and the uncapacitated plant location problem on trees. *European Journal of Operational Research 12*, 266–278.

Krarup, J. and O. Bilde (1977). Plant location, set covering and economic lot sizing: An $O(nm)$ algorithm for structured problems. In L. Collatz et al. (Ed.), *Optimierung bei graphentheoretishen und ganzzahligen Probleme*, pp. 155–180. Basel, Switzerland: Birkhauser.

Krarup, J. and P. M. Pruzan (1983). The simple plant location problem: Survey and synthesis. *European Journal of Operational Research 12*, 36–81.

Krarup, K. and P. M. Pruzan (1990). Ingredients of locational analysis. In *Discrete Location Theory*. New York: Wiley Intersicience.

Kuehn, A. A. and M. J. Hamburger (1963). A heuristic program for locating warehouses. *Management Science 9*, 643–666.

Laporte, G. (1988). Location-routing problems. In B. L. Golden and A. A. Assad (Eds.), *Vehicle Routing: Methods and Studies*, pp. 163–198. Amsterdam: North Holland.

Lawler, E. L., J. K. Lenstra, A. H. G. R. Kan, and D. B. Shymoys (Eds.) (1985). *The Traveling Salesman Problem*. Chichestor: Wiley Interscience.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Tech. J. 44*, 2245–2269.

Lin, S. and B. W. Kernighan (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21*, 498–516.

Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel (1963). An algorithm for the traveling salesman problem. *Operations Research 11*, 972–989.

Love, R. F. (1976). One dimensional facility location-allocation using dynamic

programming. *Management Science 22*, 614–617.

Love, R. F., J. G. Morris, and G. O. Wesolowsky (1988). *Facilities Location: Models and Methods*. North Holland, New York.

Lucena, A. (1990). The time-dependent tarveling salesman problem - the deliveryman case. *Networks 20*, 753–763.

Magnanti, T. L. and R. T. Wong (1981). Accelerated benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research 29*, 464–484.

Malek, K., M. Guruswamy, H. Owens, and M. Pandya (1989). Serial and parallel search techniques for the traveling salesman problem. *Annals of OR: Linkages with Artificial Intelligence*.

Maranzana, F. E. (1964). On the location of supply points to minimize transportation costs. *Operational Research Quarterly 15*, 261–270.

Menger, K. (1930). Das Botenproblem. In K. Menger (Ed.), (1932) *Ergebnisse eines Mathematischen Kolloquiums 2*, Volume 9. Kolloquium (5.II.1930), 12. Leipzig: Teubner.

Miller, C. E., A. W. Tucker, and R. A. Zemlin (1960). Integer programming formulations and traveling salesman problems. *J. Assoc. Comput. Mach. 7*, 326–329.

Miller, D. L. and J. F. Pekny (1991). Exact solution of large asymmetric traveling salesman problems. *Science 251*, 754–761.

Miller, D. L., J. F. Pekny, and G. L. Thompson (1991). An exact two-matching

based branch and bound algorithm for the symmetric traveling salesman problem. Technical report, Carnegie Mellon University, Pittsburgh.

Minieka, E. (1989). The delivery man problem on a tree network. *Annals of Operations Research 18*, 261–266.

Mirchandani, P. B. (1990). The p-median problem and generalizations. In P. B. Mirchandani and R. L. Francis (Eds.), *Discrete Location Theory*. New York: Wiley Interscience.

Mirchandani, P. B. and R. L. Francis (Eds.) (1990). *Discrete Location Theory*. New York: Wiley Interscience.

Mirchandani, P. B. and A. R. Odoni (1979). Location of medians on stochastic networks. *Transportation Science 13*, 85–97.

Mirchandani, P. B. and A. Oudjit (1980). Localizing 2-medians on probabilistic and deterministic tree networks. *Networks 10*(329–350).

Mühlenbein, H., M. Gorges-Schleuter, and O. Krämer (1988). Evolution algorithms in combinatorial optimization. *Parallel Computing 7*, 65–85.

Müller-Merbach, H. (1983). Zweimal travelling salesman. *DGOR-Bulletin 25*, 12–13.

Murty, K. G. (1968). An algorithm for ranking all the assignments in order of increasing cost. *Operations Research 16*, 682–687.

Naddef, D. and G. Rinaldi (1991). The symmetric traveling salesman polytope and its graphical relaxation: Composition of valid inequalities. *Mathematical Programming 51*, 359–400.

Naddef, D. and G. Rinaldi (1993). The graphical relaxation: A new framework for the symmetric traveling salesman polytope. *Mathematical Programming 58*, 53–88.

Narula, S. C., U. I. Ogbu, and H. M. Samuelson (1977). An algorithm for the p-median problem. *Operations Research 25*, 709–712.

Nemhauser, G. L. and L. A. Wolsey (1988). *Integer and Combinatorial Optimization*. New York: Wiley.

Norback, J. P. and R. F. Love (1977). Geometric approaches to solving the traveling salesman problem. *Management Science 23*, 1208–1223.

Norback, J. P. and R. F. Love (1979). Heuristic for the Hamiltonian path problem in Euclidian two space. *Journal of Operational Research Society 30*, 363–368.

Padberg, M. W. and M. Grötschel (1985). Polyhedral computations. In E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys (Eds.), *The Traveling Salesman Problem*, pp. 361–401. Chichester: John Wiley & Sons.

Padberg, M. W. and G. Rinaldi (1987). Optimization of a 532 city symmetric traveling salesman problem by branch and cut. *Operations Research Letters 6*, 1–7.

Padberg, M. W. and G. Rinaldi (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33*, 60–100.

Picard, J. C. and M. Queyranne (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one machine scheduling.

Operations Research 26, 86–110.

Potvin, J. Y. (1993). The traveling salesman problem: A neural network perspective. *ORSA Journal on Computing 5*, 328–348.

Potvin, J. Y. and J. M. Rousseau (1990). Enhancements to the Clarke and Wright algorithm for the traveling salesman problem. Research Report, University of Montreal.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*. Volume 840 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer-Verlag.

ReVelle, C. and R. W. Swain (1970). Central facilities location. *Geographical Analysis 2*, 30–42.

ReVelle, C. S. (1993). Facility siting and integer friendly programming. *European Journal of Operational Research 65*, 147–158.

ReVelle, C. S. and G. Laporte (1996). The plant location problem - New models and research prospects. *Operations Research 44*, 864–874.

Rosenkrantz, D. J., R. E. Stearns, and P. M. Lewis (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing 6*, 563–581.

Rosing, K. E. and P. R. Odell (1978). An application of integer programming to economic studies to offshore oil development. *Recherches Economiques de Louvain 44*, 53–69.

Rudin W. (1976). *Principles of Mathematical Analysis*. 3rd Ed. McGraw-Hill.

Sahni, S. and T. Gonzalez (1976). P-complete approximation problems. *Journal of the ACM 23*, 555–565.

Schrage, L. (1975). Implicit representation of variable upper bounds in linear programming. *Mathematical Programming Study 4*, 118–132.

Simchi-Levi, D. and O. Berman (1991). Minimizing the total flow time of n jobs on a network. *IIE Transactions 23*, 236–244.

Smith, T. H. C., V. Srinivasan, and G. L. Thompson (1977). Computational performance of three subtour elimination algorithms for solving asymmetric traveling salesman problems. *Annals of Discrete Mathematics 1*, 495–506.

Smith, T. H. C. and G. L. Thompson (1977). A lifo implicit enumeration search algorithm for the symmetric traveling salesman problem using held and karp's 1-tree relaxation. *Annals of Discrete Mathematics 1*, 479–493.

Spielberg, K. (1969). Plant location with generalized search origin. *Management Science 16*, 165–178.

Stewart, W. R. J. (1977). A computationally efficient heuristic for the traveling salesman problem. In *Proc. 13th Annual Meeting of S. E. TIMS*, pp. 75–85.

Stollsteimer, J. F. (1963). A working model for plant numbers and locations. *Journal of Farm Economics 45*, 631–645.

Tansel, B. Ç., R. L. Francis, and T. J. Lowe (1983). Location on networks: A survey. *Management Science 29*, 482–511.

Teitz, M. B. and P. Bart (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research 16*, 955–961.

Thisse, J. F. and H. G. Zoller (Eds.) (1983). *Locational Analysis of Public Facilities.* North Holland, Amsterdam.

Tsitsiklis, J. (1992). Special cases of the traveling salesman and repairman problems with time windows. *Networks 22*, 263–282.

Ulder, N. L. J., E. Pesch, P. J. M. van Laarhoven, H. J. Bandelt, and E. H. L. Aarts (1990). Improving TSP exchange heuristics by population genetics. Technical report, Erasmus Universiteit, Rotterdam.

Vander Wiel, R. J. and N. V. Sahinidis (1996). An exact solution approach for the time-dependent traveling-salesman problem. *Naval Research Logistics 43*, 797–820.

Vandermonde, A. T. (1771). Remarques sur les problèmes de situation. *Histoire de l'Académie des Sciences(Paris)*, 566–574.

Voigt, B. F. (1831). *Der Handlungreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen in seinen Geschäften gewiss zu sein.* Ilmenau: Von einem alten Commis-Voyageur. (Republished (1981) Verlag Bernd Schramm, Kiel.

Volgenant, T. and R. Jonker (1982). A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research 9*, 83–89.

Weber, A. (1909). *Über den Standort der Industrien: Erster Teil, Rein Theorie des Standorts*, J. C. B. Mohr.

Wesolowsky, G. O. (1993). The Weber problem: History and perspectives. *Location*

Science 1, 5–23.

Wiorkowski, J. J. and K. McElvain (1975). A rapid heuristic algorithm for the approximate solution of the traveling salesman problem. *Transportation Research 9*, 181–185.