TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMARBEIT

Facility Location and Allocation Problems with Stochastic Customer Demand and Immobile Servers

Ausgeführt am Institut für
Wirtschaftsmathematik
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gernot Tragler

durch

Stefanie Stiermaier
Waldrandsiedlung 117
3910 Zwettl

Wien, Mai 2010

Acknowledgments

This thesis and all my years of study would not have been feasible without the support and encouragement of several people.

First and foremost, I want to express my sincerest gratitude to my advisor Gernot Tragler, who introduced me to Operations Research and inspired my interest in this area. He gave me the opportunity to write my thesis on this topic. I want to thank Gernot Tragler for his guidance, valuable suggestions, patience and sharing of knowledge.

Secondly, I am very grateful to Robert Aboolian from California State University, San Marcos and Zvi Drezner from California State University, Fullerton for their cooperation, their friendly support and for very valuable advices on my thesis.

Special thanks to my boyfriend Martin, who supported me in implementing the developed algorithms. Thank you for your dedication, your selfless devotion and your endless support in all matters.

I would also like to thank my friends and fellow graduate students for always being there for me. In particular, I am very thankful to Maria. She took the time to read parts of my thesis and provided useful hints.

Last but not least, I am much obliged to my parents for their financial support through my study, for always keeping me motivated and for their faith.

Abstract

In this thesis the problem of the optimal location of service facilities in the presence of stochastic demand is analyzed. Fixed servers are allocated to the facility sites. Customers generate demand for service at each node of the network. The demand rate is assumed to be Poisson distributed. The customers travel to the closest facility to obtain service and thus generate the arrival rate at the stations. There is no upper limit on the capacity of the facility nodes. The established servers provide service according to a certain service rate. The objective is to find the optimal number of facility sites, their location and the number of assigned servers in order to minimize the cost of the system.

Motivating applications for this optimization problem include the location of warehouses and stores, as well as walk-in health clinics.

In particular, this thesis considers two different location models. In the first one, the Multiple Server Problem, a given number of servers is to be located in order to minimize the sum of total travel and waiting times in the system. Contrary to this, the Total Cost Model does not limit the number of server units. The total cost of the system including travel and waiting time of all customers, fixed installation costs for facilities and variable server costs is to be minimized.

Both problems are formulated and analyzed. Heuristics, such as Simulated Annealing, Genetic Algorithm, and Tabu Search are introduced and implemented. A sensitivity analysis is carried out.

The computational results show that the problems can be solved efficiently by using the Genetic Algorithm. Not only does it detect the best solution for most of the problems, but also the results obrained are always among the best of all heuristics. Therefore, the Genetic Algorithm is recommended for solving the underlying Stochastic Location Models.

Kurzfassung

In dieser Arbeit wird das Problem der optimalen Positionierung von Dienstleistungseinrichtungen mit stochastischer Nachfrage analysiert. Fixes Personal wird den Standorten zugewiesen und leistet Service gemäß einer gegebenen Servicerate. Die Kunden sind bei den Knoten eines Netzwerkes platziert, und ihre Nachfrage ist Poisson-verteilt. Um bedient zu werden, kommen sie zu der nächstgelegenen Geschäftsstelle. Somit erzeugen sie die Ankunftsrate bei den jeweiligen Standorten. Ziel ist es, die optimale Anzahl an Geschäftsstellen, deren Position und die optimale Anzahl an zugewiesenem Personal für jede Position zu finden, sodass die Kosten des Systems minimiert werden.

Anwendungsbeispiele dieses Optimierungsproblems beinhalten die Positionierung von Warenlagern, Filialen, Ambulanzen oder Polizeistationen.

Im Besonderen werden in dieser Arbeit zwei Modelle betrachtet. Im "Multiple Server Problem" wird eine gegebene Anzahl an Personal, das zur Verfügung steht, so plaziert, dass die Summe der Reise- und Wartezeiten aller Kunden minimal ist. Im Gegensatz dazu schränkt das "Total Cost Problem" die Gesamtanzahl des Personals nicht ein. Die Zielfunktion besteht aus den Gesamtkosten des Systems, die sich aus Reise- und Wartezeiten der Kunden, Errichtungskosten für die Geschäftsstellen und Personalkosten zusammensetzen.

Beide Probleme werden formuliert und analysiert. Heuristiken wie Simulated Annealing, Genetische Algorithmen und Tabu Search werden erklärt und implementiert. Anschließend wird eine Sensitivitätsanalyse durchgeführt.

Die numerischen Ergebnisse zeigen, dass beide Modelle mit dem Genetischen Algorithmus auf eine effiziente Art gelöst werden können. Er liefert nicht nur die beste Lösung für einen Großteil der Probleme, sondern die Ergebnisse haben auch die geringsten Abweichungen von dem besten gefundenen Resultat.

Contents

Chapter 1

Introduction

Facility Location Problems aim to find the optimal position of service facilities in order to minimize the total cost or to maximize the total coverage of the system. From the customer's point of view, there are two key issues that should be considered when choosing the location: the traveling distance to the facilities and the expected waiting times at the service sites.

Therefore, in this thesis, optimization problems with two decision variables are studied: the location of the facilities and the capacity of each service site. The facilities act as queuing systems and the objective is to minimize the total cost of the system.

The demand rate of the customers and the service rate are considered. Since these two values are stochastic, an interplay of location and stochasticity arises. Such Stochastic Facility Location Models are very complicated and thus rather strong assumptions are made to allow for a reasonable analysis: Poisson arrival and service rates, a fixed number of servers, and a discrete set of possible facility sites. Furthermore, customers are assumed to travel to the closest facility to obtain service.

In this thesis, optimization models for the following two problems are presented and solved:

1. The Multiple Server Problem: The total consumer costs consisting of travel costs and waiting costs are to be minimized. An upper bound on the number of assigned servers is given.
2. The Total Cost Problem: Besides travel and waiting costs, fixed costs for opening facilities and variable costs for installing servers are considered. The number of servers is not limited.

The applications for Stochastic Facility Location Problems range from where to locate the offices in public facilities such as emergency services, police stations, or government offices to private facilities such as warehouses, stores, or bank offices.

This thesis is organized as follows:

In Chapter 2 a detailed introduction to Facility Location Problems with Uncertainties is given. After discussing the underlying model, the differences and similarities between the main model types are outlined. Moreover, a short literature review is presented.

Chapter 3 analyzes the trade-off between several components of the objective function. The Multiple Server Problem and the Total Cost Problem are formulated and differences are discussed. Finally, an algorithm to obtain the optimal assignment of servers is developed for both problems.

Heuristics to find the optimal set of facility locations are outlined in Chapter 4. After defining the neighborhood of a solution set and giving a short classification of heuristics, the Descent Algorithm is formulated. The three guided random search techniques Simulated Annealing, Genetic Algorithm, and Tabu Search are discussed. For each of them, a short introduction is given before formulating the algorithms for the underlying problems.

The four solution procedures from the preceding chapter are implemented for several problems. The computational results are reported in Chapter 5. The quality of their results is considered and they are analyzed. Additionally, in the second part of the experiments, a sensitivity analysis using one of the heuristics is presented.

Finally, Chapter 6 summarizes the most important results of this thesis and provides a proposal for further research concerning Stochastic Facility Location Problems.

Chapter 2

Stochastic Location Models

2.1 Location Problems with Stochastic Demand and Congestion

Facility Location, sometimes also called Location Analysis, is concerned with the placement of one or more facilities by choosing from a set of possible positions. Certain objectives, such as minimizing costs or maximizing the covered population, have to be considered. The problem is to find an optimal location for facilities while respecting effort (e.g. costs) and utility (e.g. maintenance of service).

In 1909, when Alfred Weber tried to find an optimal position for a warehouse, the study of Facility Location began. Since not only mathematicians are attracted by this problem, Facility Analysis is a well established research area and a lot of literature on this topic exists (for a literature survey, see Felsenstein and Stiermaier, 2009).

If uncertainties in finding the optimal location of a facility occur, this problem is referred to as a Stochastic Location Model. Mainly there are two sources of uncertainty:

1. Stochastic customer demand: The exact time and amount of customer demand generated at different locations are stochastic variables.
2. Potential congestion at the service facility: The quantity of the customers, who access service at a certain service station, is also a stochastic variable.

This class of location problems is referred to as "Location Problems with Stochastic Demand and Congestion" in Berman and Krass (2002).

In the mid 1970s, Richard C. Larson started the research on Queueing-Location Problems (see Larson, 1974). Subsequently, many researchers concentrated on location problems where queueing occurs.

At certain locations, facilities provide service. The clients come or are taken to these fixed stations. Their arrival rate is assumed to be Poisson distributed and they are served according to a certain service rate. If all resources at a facility are exhausted at a time, the clients have to wait in a queue and are served as soon as a service unit becomes available.

In order to avoid such queues, not only the optimal locations for the service stations are sought, but also the best service capacity, i.e., the quantity of servers allocated at each service facility. Thus, we have two decision variables, the nodes of the service facilities and the number of assigned servers.

A huge number of objectives in Facility Location exists. The producers seek profit maximization either by minimizing costs or maximizing demand. From the customer's point of view, cost minimization is the main objective. And for others, environmental concerns are the most important ones.

Since it is assumed that customers are free to choose the facility, congestion plays an important role in these models. If clients anticipate encountering a long queue at the closest facility, they might choose another one.

Applications for Facility Location models with Stochastic Demand arise in public as well as in private sectors. The most popular one is the location of emergency service facilities such as hospitals, police stations, or fire stations. But also for retail outlets, Facility Location Models play a crucial role.

2.2 General Model Description

Let $G = (N, L)$ be a network, where N is a set of nodes ($|N| = n$) and L is a set of links. d_{ij} denotes the shortest distance between node i and node j ($i, j \in N$).

2.2.1 Components

According to Berman and Krass (2002), there are four main components in Queueing-Locations Problems:

- Customers: They are located at each node $i \in N$ ($|N| = n$) and generate demand at a rate of λ_i per unit of time at node i ($\sum_{i \in N} \lambda_i = 1$). It is typically assumed that the demand rate is Poisson distributed.[1]

- Facilities: Let $M \subset N$ ($|M| = m$) be a set of potential locations for the facilities. Thus, at most m facilities can be placed on the network.

Figure 2.1 illustrates a sample network of customer and facility nodes. In this case, $|N| = 5$ and $|M| = 2$. It clarifies that the service nodes are a subset of the customer nodes.

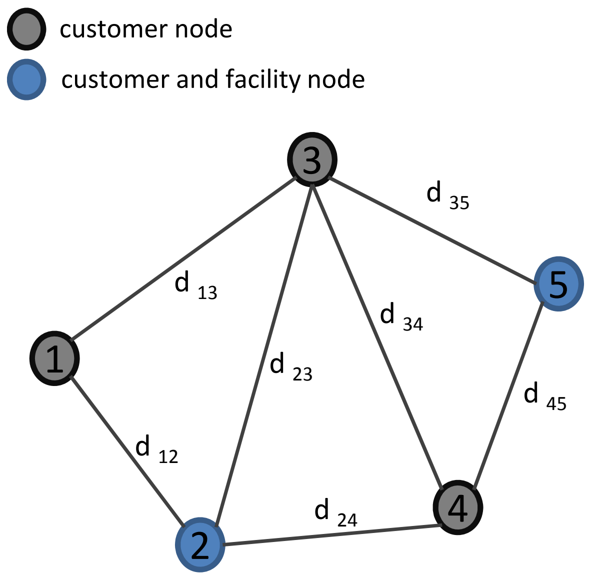

Figure 2.1: Sample Network

[1]The Poisson distribution can be derived as the limit of a binomial distribution. X is Poisson distributed with parameter λ, if $P(X = n) = \frac{\lambda^n e^{-\lambda}}{n!}$. The mean of a Poisson distribution is $E(X) = \lambda$ (see, e.g., Hillier and Lieberman, 2005).

- Servers: For simplicity it is assumed that all servers have the same service rate of μ customers per unit time. Just like the demand rate, the service rate is usually Poisson distributed. Let k_j be the quantity of servers located at station j and p the maximum number of servers that can be allocated to all facilities ($\sum_{j \in M} k_j \leq p$). A node $j \in M$ may have no server or any positive integer number less or equal to p. The servers can either be fixed at the facility location or mobile (travel to the customers).
- Calls for Service: When a customer calls for service, it first must be determined that the call comes from a "covered" node. If the standards for coverage are not fulfilled, a penalty has to be paid. A "covered" customer is placed in a queue and served once a server becomes available. The total response time of a customer consists of the waiting time and the travel time.

The queueing system at a particular facility node is shown in Figure 2.2. λ_j customers come to the facility site, where k_j servers are placed. After being served, the customers leave the queueing system.

In addition to these four above-mentioned components, further assumptions are required for formulating a Stochastic Location Model. Among others, the decision variables, the objective function, and the nature of the servers have to be determined.

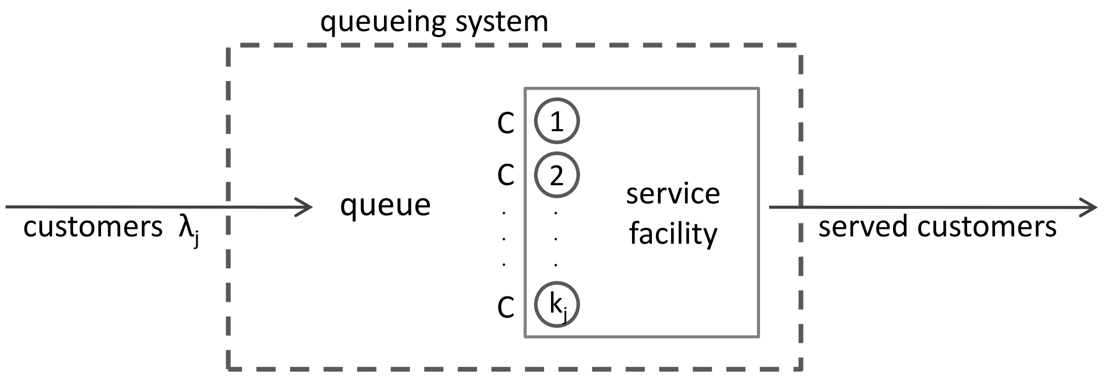

Figure 2.2: Queueing System at Facility Node j

2.2.2 Decision Variables

First, we have to decide where to locate the facilities and how many servers to place there. Therefore, a facility location vector $\mathbf{x}$ is defined.

$$x_j = \left\{ \begin{array}{c} k \\ 0 \end{array} \right\}, \text{ if} \left\{ \begin{array}{l} \text{a facility with } k \text{ servers} \\ \text{no facility} \end{array} \right\} \text{ is located at } j.$$

The second decision variable $\mathbf{y}$ is called coverage vector. This binary vector provides information about whether or not a customer node is covered by one of the service facilities.

$$y_i = \left\{ \begin{array}{c} 1 \\ 0 \end{array} \right\}, \text{ if} \left\{ \begin{array}{l} \text{demand point } i \text{ is covered by a facility} \\ \text{otherwise} \end{array} \right\}.$$

Later on, x_j will also be a binary variable and a new vector $\mathbf{K}$ denoting the assigned number of servers to each facility will be defined. Furthermore, $\mathbf{y}$ will indicate if customers from node i are served by facility j (see Section 3.2).

2.2.3 Components of the Objective Function

The objective function consists of four components that should be minimized. As already mentioned above, penalty costs have to be paid for clients, who are not covered by one of the facilities. Therefore, the total cost of non-covering customers NC has to be considered. But even if a client is covered, it cannot be guaranteed that there is enough space in the queue. Thus, rejection costs RC are the second component of the objective function. And finally, the total waiting costs WC of all customers and the costs for installing servers IC at the facilities should also be as small as possible.

Consequently, the total cost TC of the system is

$$TC = NC + RC + WC + IC.$$

In detail, the costs can be calculated as follows:

- Let c_{NC} be the penalty for each non-covered client. Remember that λ_i denotes the demand rate at node i. The total cost of non-covering is given by

$$NC = \sum_{i \in N} c_{NC} \lambda_i (1 - y_i).$$

- For each rejected customer, a penalty of c_R has to be paid. Considering the probability $P_R^i(\mathbf{x},\mathbf{y})$ of rejecting a client from node i, the total cost of rejection is

$$RC = \sum_{i \in N} c_R P_R^i(\mathbf{x},\mathbf{y}) \lambda_i y_i.$$

- The waiting time for a customer from node i consists of the expected time $W_i(\mathbf{x},\mathbf{y})$ spent in the system and the expected travel time. Let c_W be the waiting costs per unit time. $P_D^{ij}(\mathbf{x},\mathbf{y})$ is the probability that a client at node i is served by facility j. The shortest distance between i and j is denoted by d_{ij} and v is assumed to be the average travel speed. Then the total waiting costs are given by

$$WC = \sum_{i \in N} c_W (1 - P_R^i(\mathbf{x},\mathbf{y})) \lambda_i y_i \left[W_i(\mathbf{x},\mathbf{y}) + \frac{1}{v} \sum_{j \in M} P_D^{ij}(\mathbf{x},\mathbf{y}) d_{ij} \right].$$

- For locating k servers at location j, c_L^{jk} has to be paid. Recall that the upper bound on the number of servers allocated to all facilities is p. The total installing costs are therefore

$$IC = \sum_{j \in M} \sum_{k=1}^{p} c_L^{jk} \mathbb{1}_{\{x_j = k\}}.$$

2.2.4 Constraints

There are several requirements on the decision variables that have to be considered and fulfilled.

Limited Number of Servers
The maximum number of servers allocated to the facilities is given by p. Thus, the first constraint is

$$\sum_{j \in M} k_j \leq p.$$

Limited Number of Facilities
Since m is the upper limit on the total number of facilities, the second condition is obtained

$$\sum_{j \in M} \mathbb{1}_{\{x_j > 0\}} \leq m.$$

Desired Bound on the Response Time

The third constraint is a probabilistic one that describes a desired bound on the response time of a customer request.

In case of emergency service, it is of particular importance that the response time to a call is as short as possible. For example, it might be required that at least 95% of all calls have a response time less than three minutes.

Let u_i be the lower bound on the service level (in our example: three minutes) and $SL_i(\mathbf{x}, \mathbf{y})$ the random service level for a customer at node i (e.g. response time). α_i denotes the minimum desired frequency (e.g. 95%).

For the optimal solution, the probability that the service level is smaller than the required bound u_i has to be larger than the desired frequency at all service stations.

$$P\left(SL_i(\mathbf{x}, \mathbf{y}) \le u_i\right) \ge \alpha_i y_i, \quad \forall i \in N. \tag{2.1}$$

2.2.5 General Model Formulation

Combining our results from the preceding Sections 2.2.2, 2.2.3, and 2.2.4, the general model for Facility Location with Stochastic Demand can now be formulated as follows:

$$\begin{aligned}
\min\ & TC = NC + RC + WC + IC & & \quad (2.2)\\
\text{s.t.} & \sum_{j \in M} k_j \le p & & \\
& \sum_{j \in M} \mathbb{1}_{\{x_j > 0\}} \le m & & \\
& P\left(SL_i(\mathbf{x}, \mathbf{y}) \le u_i\right) \ge \alpha_i y_i, & & \forall i \in N,\\
& x_j \in \{0, \dots, p\}, & & \forall j \in M,\\
& y_i \in \{0, 1\}, & & \forall i \in N.
\end{aligned}$$

At this point we would like to emphasize that problem (2.2) provides quite a general formulation. In what follows, several modifications especially of the constraints will be considered.

2.3 Different Types of Models

2.3.1 Classes of Models

Considering the objective function, we differentiate between Coverage-Type Models and Median-Type Models.

Coverage-Type Models
As their name already denotes, the main objective of Coverage-Type Models is to provide adequate coverage for each customer, i.e., at least one service station within a given maximal distance of each client. In most of these models, the travel costs are not considered in the objective function.

Berman and Krass (2002) divide the Coverage Models into the following two classes: The first one is the Set Cover-Type Problem, where the focus is on minimizing the total number of installed facilities. Only the total cost for installing servers IC remains in the objective function and thus it becomes linear. However, the non-linearities in the constraints remain. Contrary to these models, the Maximal Weight Cover Problems aim to maximize the total demand (weight) of all clients. The only component in the objective function is the cost of non-covering customers NC.

Median-Type Models
Median-Type Models usually assume a fixed number of facilities, and clients seek service from the closest service station, where all of them are covered by one of the facilities. Thus, variable $\mathbf{y}$ and the third constraint (2.1) can be dropped. The objective is to minimize the travel and waiting costs. The following non-linear optimization problem in general can only be solved by heuristics.

$$
\begin{aligned}
\min\ & RC + WC \\
\text{s.t.} & \sum_{j \in M} k_j = p \\
& \sum_{j \in M} \mathbb{1}_{\{x_j > 0\}} = m \\
& x_j \in \{0, \ldots, p\}, \qquad \forall j \in M.
\end{aligned}
$$

Assuming that customers have full information about the waiting time at the stations leads to very difficult formulations. The clients will choose the facility that minimizes travel and waiting time. Thus, demand becomes a decreasing function in the expected waiting time.

2.3.2 The Nature of the Servers

Median-Type Models as well as Coverage-Type Models can be classified into two categories: models with fixed servers and models with mobile servers.

In the case of fixed facilities, the customers travel to the service station and obtain service there. Mobile servers travel from their home facility to the customer's location. After providing the required on-scene service, the server returns (with or without the customer) to the station. If the customers are taken to the facility site, they are served there (e.g. patients are provided first aid by ambulance men and emergency doctors on scene and are taken to the hospital for further treatment). Thus, for mobile server units the travel time to the customer and back to their home facility is part of the service time. While in the case of fixed servers, it is part of the travel time.

Illustrative Example

The following example (see Berman and Krass, 2002) points out that there are fundamental differences between these two categories.

A system with only one facility is considered. The facility houses k servers and has a queueing capacity of one customer. The service rate is assumed to be exponentially distributed.[2] A fixed service facility operates as $M/M/k/1$ queueing system,[3] for which analytical results are available (e.g. exact formulas for the expected waiting time). Therefore, an analytical formulation of the model can be obtained.

In the mobile server case, the service is also supposed to be exponentially distributed. Travel times are assumed to be deterministic. The total service time consists of the travel time to the customer and back to the facility. Since travel times are not exponentially distributed in this case, we are dealing with an $M/G/k/1$ system. The analytical formulation cannot be written down, because analytical results are unavailable. Thus, models with mobile servers nearly always require a combination of approximations and heuristic solution procedures.

[2]A random variable is said to be exponentially distributed with parameter λ if its probability density function is $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$. The expected value is given by $E(X) = \frac{1}{\lambda}$ (see, e.g., Hillier and Lieberman, 2005).

[3]Kendall's Notation for Queuing Systems: A/B/X/Y, where A denotes the arrival time distribution, B the service time distribution, X the number of service units, and Y the queueing capacity; G stands for general (i.e., not specified), M for Markovian (exponential), and D for deterministic (see, e.g., Hillier and Lieberman, 2005).

As explained above, the underlying system performance characteristics change drastically depending on whether servers are fixed or mobile. Therefore, a fundamental difference between the models is in the nature of servers.

It is clear that emergency stations (such as fire, police, or ambulance stations) have mobile servers and strong coverage constraints. Models developed for non-emergency systems typically have fixed servers and assume that all customers are covered.

In Figure 2.3 the classification of Stochastic Location Models is summarized. However, to make it easier to read, the nature of the servers for Coverage Type Models is not included.

In this work Median-Type Models with fixed servers are considered. Customers are assumed to travel to the closest service facility and obtain service there. Applications for our formulations can be found in the placement of Automatic Teller Machines (ATM), the location of bank branches and post offices, or any other similar service business.

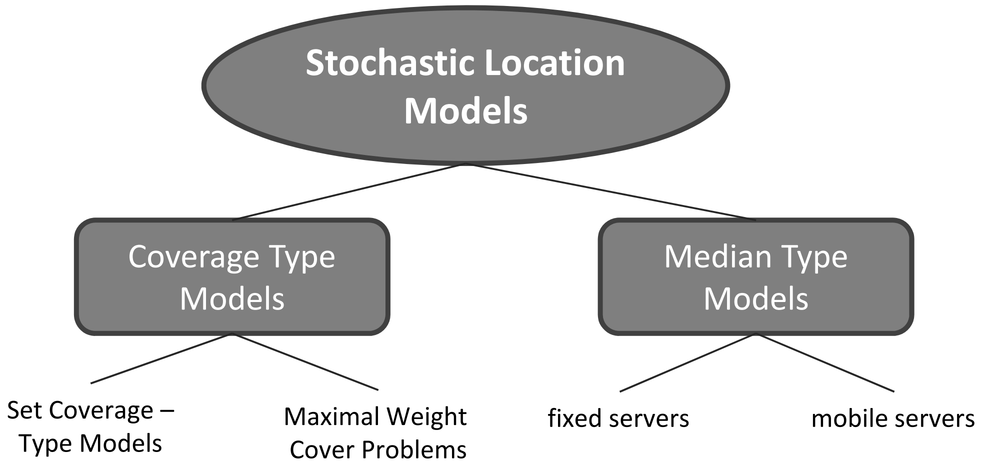
Stochastic Location Models
Coverage Type Models
Median Type Models
Set Coverage – Type Models
Maximal Weight Cover Problems
fixed servers
mobile servers

Figure 2.3: Classification of Stochastic Location Models

2.4 Literature Review

2.4.1 Median-Type Models

Median-Type Models with Mobile Servers

Primarily the development of Facility Location Models with Stochastic Demand and Congestion has been motivated by applications of emergency service stations. The calls for service are stochastic. Mobile servers such as ambulances or fire engines travel from their station to the site of the emergency.

Berman et al. (1985) first discuss the Stochastic Queue Median (SQM) with a single server. They assume an $M/G/1$ queueing system and First-Come-First-Served manner.

In realistic emergency models, however, not all calls have the same priority. Therefore, Batta et al. (1988) extend the SQM and consider the problem of different priority classes. The authors found out that the optimal location is not the same.
Batta and Berman (1989) discuss the SQM with k servers at a single location. As this model is seen as an $M/G/k$ queue, an adequate approximation for the waiting time is used.

Other extensions can be found in Berman and Mandowsky (1986) and Berman et al. (1987). The focus is on finding the best location for a certain number of facilities with one mobile server at each station. Furthermore, Berman and Mandowsky allow cooperation between the servers.

Median-Type Models with Fixed Servers

Recent papers on Median-Type Models concentrate on problems with fixed servers, where customers travel to the facilities to obtain the required service. Most of the authors assume that clients do not have full information and hence seek service at the closest facility.

Wang et al. (2004) consider an $M/M/1$ queueing system. Only one server can be placed at a service station. The costs of the system (i.e., travel time and waiting time of the customers) should be minimized. There are constraints on the maximum expected waiting time.

A greedy-dropping heuristic, a Tabu Search, and a Lagrangian Relaxation for the latter optimization problem are developed in Wang et al. (2002).

The focus in Berman et al. (2006) is on minimizing the total number of service stations. Additionally, the authors assume that customer demand is lost if certain constraints on congestion and coverage are not fulfilled.

The Multiple Server Location Problem is introduced by Berman and Drezner (2007). At most k servers are allowed to be placed at any potential service station. The location of the facilities and the quantity of server units are decision variables. The objective of this model is to minimize the sum of travel time and waiting time at the server. Several heuristics are presented in this paper to solve the optimization problem.

A minimax-type model of this problem is presented in Aboolian et al. (2009), where the focus is on minimizing the maximum time spent in the system (i.e., the sum of travel time and waiting time of each customer). A Descent and a Genetic Algorithm are developed to find the optimal facility sites.

Aboolian et al. (2008) discuss a problem that differs in several assumptions from the models mentioned so far. First of all, the objective is to minimize the total cost of the system, which includes not only the travel and waiting time of all customers, but also the fixed installation costs of facilities and the variable costs of servers. There is no limit on the quantity of facilities and servers. In addition to heuristic approaches (Descent and Simulated Annealing Algorithm), an exact solution algorithm is developed.

Finally, Drezner et al. (2010) consider an extension of Berman and Drezner (2007), where customers do not seek service at the closest facility. The gravity rule is used to describe the demand rates of the different stations. In the first model, the Stationary Model, the customers are assumed to neglect the expected waiting times at the facility. They act according to the gravity rule. In contrast to these assumptions, the clients in the Interactive Model know and consider the waiting time when choosing the facility. Hence, the second problem is more difficult to analyze. For solving these two location problems heuristics (Descent Algorithm, Simulated Annealing, and Tabu Search) are developed.

2.4.2 Coverage-Type Models

It is not surprising that a huge amount of research concerning Coverage-Type Models exists. As before, this review is divided into models with mobile servers and models with fixed servers.

Coverage-Type Models with Mobile Servers

The first model in the field of Coverage-Type Models with mobile servers is developed by Daskin (1983), who considers the Maximum Expected Covering Location Problem (MECLP), where three simplifying assumptions concerning the servers are made. Batta et al. (1989) revisit Daskin's model and relax these assumptions.

The Maximum Availability Location Problem (MALP) by ReVelle and Hogan (1989) assumes that each location can house exactly one server unit.

Also in Marianov and ReVelle (1994) and Marianov and ReVelle (1996) each facility is assumed to contain only one server. The Queueing Probabilistic Location Set Covering Problem focuses on minimizing the total number of server units (see Marianov and ReVelle, 1994). The authors also formulate the Queueing-Maximum Availability Location Problem, where it is assumed that each neighborhood can be treated as separate queueing system and that the travel times to customers are much smaller than the service times (see Marianov and ReVelle, 1996).

Coverage-Type Models with Fixed Servers

The research on Coverage-Type Models, where customers travel to the facilities in order to obtain service, started later. Marianov and Serra (1998) describe the problem of locating M fixed facilities and allocating the users to them. The objective is to maximize the total coverage and to capture as much demand as possible. Two different assumptions (one server at each facility and k servers at each facility) are made. The number of servers is fixed in advance and identical for all stations. Each service facility thus represents an $M/M/1$ or $M/M/k$ queue with finite capacity.

In contrast to these formulations, the total number of facilities is part of the decision in Marianov and Ríos (2000) formulation. Also the quantity of servers at each site is not fixed in advance, but rather a decision variable.

Chapter 3

The Multiple Server Problem and the Total Cost Problem

For a given network, the problem of locating facility stations at some nodes and allocating servers to these sites is considered. Customers are placed at the nodes. Since the servers are assumed to be fixed and the customers have to travel to the facilities, they select the closest station (i.e., the distance between their 'home' node and the chosen facility is minimized). Due to certain constraints, a facility cannot be located at every node. Therefore, a subset of nodes for potential facility sites needs to be selected. The number of servers to be established there can vary from zero to any positive integer number. As a result, the facilities act as $M/M/k$ queueing systems (with k denoting the number of servers).

The objective is to minimize the sum of total cost for all customers (the time of traveling to the closest station and the average time spent at the facility itself) and the total cost of establishing the facilities (the costs of opening facilities and the costs for the servers).

The area of applications for this kind of problems is widely spread. Generally, they can be applied for any establishment that serves customers and where the number of servers is a decision variable. The most popular example mentioned in the literature on Facility Location is the location of Automatic Teller Machines (ATM) (see, e.g., Berman and Drezner, 2007). But also the location of bank branches, post offices, or similar establishments exemplifies this kind of problems.

3.1 On the Pooling of Services

In the objective function there is a negative trade-off between several components. In particular, the travel costs and the waiting costs are considered.

On the one hand, if serving units are located at as many facilities as possible, the total travel costs are reduced. The installation costs and the waiting costs, however, increase.

On the other hand, reducing the quantity of facilities and pooling servers at the selected sites lowers the time spent in the queue. It is obvious that the demand at the stations increases. Nevertheless, it can be shown that the waiting times are reduced (see Drezner, 1988).

The arrival rate λ is assumed to be Poisson distributed, the service rate μ exponentially distributed. The number of servers in the system is denoted by k. The formulae of the probability that no server is in the system, P_0, of the average number of customers in the queue, L_q, and in the system, L, and of the average time in the queue, W_q, and in the system, W, for the $M/M/k$ case are given by (see, e.g., Hillier and Lieberman, 2005)

$$P_0 = \left[\sum_{n=0}^{k-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} + \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} \frac{1}{1-\rho} \right]^{-1}, \tag{3.1}$$

$$L_q = \frac{P_0 \left(\frac{\lambda}{\mu}\right)^k \rho}{k!\,(1-\rho)^2},$$

$$L = L_q + \frac{\lambda}{\mu},$$

$$W_q = \frac{L_q}{\lambda}, \quad \text{and} \tag{3.2}$$

$$W = W_q + \frac{1}{\mu}, \tag{3.3}$$

where $\rho = \frac{\lambda}{k\mu} < 1$, the utilization of all server units.

For an efficient way of calculating W and W_q see Appendix A.1. In the following, these results are used for all waiting time calculations.

Assuming that s stations (s being a positive even integer) each served by only one service unit are given, leads to an average waiting time in the system of $W(1) = \frac{1}{\mu-\lambda}$ and to an average waiting time in the queue of $W_q(1) = \frac{\lambda}{\mu(\mu-\lambda)}$ at each station.

If only $s/2$ facilities are opened, but two servers are placed at each station, the arrival rate is doubled (i.e., 2λ). The waiting times are reduced to $W(2) = \frac{\mu}{\mu^2-\lambda^2}$ and $W_q(2) = \frac{\lambda^2}{\mu(\mu^2-\lambda^2)}$.

In Drezner (1988) it is shown that the ratio between $W(2)$ and $W(1)$ is always less than one (i.e., the waiting time when two servers are placed at $s/2$ station is smaller). This is also the case for the ratios of the waiting times in the queue and the ratios of the numbers of customers in the queue. Only the number of clients in the system does not decrease if two servers are located at $s/2$ facilities ($L(2) > L(1)$). This follows from the fact that two customers can be served at the same time.

$$\begin{aligned}
\frac{W(2)}{W(1)} &= \frac{\mu}{\mu+\lambda} < 1 \\
\frac{W_q(2)}{W_q(1)} &= \frac{\lambda}{\mu+\lambda} < 1 \\
\frac{L_q(2)}{L_q(1)} &= \frac{2\lambda}{\mu+\lambda} < 1 \\
\frac{L(2)}{L(1)} &= \frac{2\mu}{\mu+\lambda} > 1
\end{aligned}$$

Is it really possible to improve the situation without paying more? Does the pooling of servers lead to a better solution without additional costs?

Drezner (1988) supposed that the servers might be exploited more. Using the following arguments, he showed that this is not the case.

If only one server is placed at a station, the probability of no customer being in the system is given by $P_0(1) = 1 - \frac{\lambda}{\mu}$. In the two server case, $P_0(2) = \frac{2\mu-\lambda}{2\mu+\lambda}$. The percentage that only one client is in the system and thus just one server unit is busy is $P_1(2) = \frac{\lambda}{\mu}P_0(2) = \frac{\lambda(2\mu-\lambda)}{\mu(2\mu+\lambda)}$. Therefore, the probability that one server is idle can be calculated by $P_1 = P_0(2) + \frac{1}{2}P_1(2) = 1 - \frac{\lambda}{2\mu}$, which is the same as $P_0(1)$. (The service rate is doubled since two servers are at $s/2$ stations.)

We have seen that the situation can be improved just by pooling servers.

3.2 Model Formulation

Even if two different models are discussed in this chapter, i.e. the Multiple Server Problem and the Total Cost Problem, we start with a general formulation here and go into detail later in the next section.

As already mentioned in the General Model Description (see Section 2.2), a graph $G(L, N)$ is needed. N denotes a set of $|N| = n$ nodes, where the customers are located and demand rate λ_i is generated at each node $i \in N$. L represents the link set. The shortest path between the nodes i and j $(i, j \in N)$ is given by d_{ij}. It is assumed that each server has the same service rate of μ customers and a maximum of p servers are to be located in the system. $M \subset N$ $(|M| = m)$ denotes the potential facility locations and $S \subset M$ the set of the selected nodes, where facility sites are placed.

The facility location vector $\mathbf{x}$ (see Section 2.2.2) is now a binary variable and its components are defined as follows:

$$x_j = \left\{ \begin{array}{c} 1 \\ 0 \end{array} \right\}, \text{ if} \left\{ \begin{array}{l} \text{a service site is placed at node } j \in M \\ \text{otherwise} \end{array} \right\}. \tag{3.4}$$

The coverage vector $\mathbf{y}$ is given by $(i \in N$ and $j \in M)$

$$y_{ij} = \left\{ \begin{array}{c} 1 \\ 0 \end{array} \right\}, \text{ if} \left\{ \begin{array}{l} \text{demand node } i \text{ is covered by facility } j \\ \text{otherwise} \end{array} \right\}. \tag{3.5}$$

Since these two decision variables do not provide any information about the number of allocated servers at each facility, a third variable is needed (see Aboolian et al., 2008). Let $\mathbf{K}(S) = \{k_j(S) | j \in S\}$ be the assignment vector. The components $k_j(S)$ denote the quantity of server units located at facility $j \in S$. Due to the fact that the total number of servers is limited to p, each component $k_j(S)$ is an integer between 0 and p.

For each node $j \in S$, we define $C_j(S)$, the set of all customer nodes that choose facility j (i.e., all client nodes, for which the distance to service site $j \in S$ is the smallest compared to all other sites in S). Therefore, the arrival rate at facility j is given by

$$\lambda_j(S) = \sum_{i \in C_j(S)} \lambda_i.$$

The average waiting time in the queue, W_q, at a service station depends on the arrival rate λ, the service rate μ and the number of servers k at the station. It is written as $W_q(\lambda, \mu, k)$ and characterized by the following two properties:

Property 1: $W_q(\lambda, \mu, k)$ is a decreasing function in the number of servers k.

Property 2: $W_q(\lambda, \mu, k)$ is a convex function in the number of servers k.

PROOF: see Appendix A.2.

The focus is now on minimizing the total cost of the system, which consist of travel, waiting, installation, and server costs.

Let g be the travel costs per unit of distance. Since d_{ij} denotes the distance between node i and facility j, the total travel costs of all customers in the system are

$$TC = g \sum_{j \in S} \sum_{i \in C_j(S)} \lambda_i d_{ij}.$$

The waiting costs per unit of time are assumed to be v. Therefore, the expected total costs of waiting equal

$$WC = v \sum_{j \in S} \lambda_j(S) W(\lambda_j(S), \mu, k_j(S)),$$

where $\lambda_j(S) = \sum_{i \in C_j(S)} \lambda_i$.

The costs of opening facilities consist of the fixed installation costs f_j for facility $j \in S$ and variable server costs h per server.

$$OC = \sum_{j \in S} f_j + h \sum_{j \in S} k_j(S)$$

is thus the total cost of opening service sites.

Then, the objective function is formulated as follows:

$$F(S, \mathbf{K}(S)) = \sum_{j \in S} \left\{ g \sum_{i \in C_j(S)} \lambda_i d_{ij} + v \lambda_j(S) W(\lambda_j(S), \mu, k_j(S)) + \right. \\ \left. + f_j + h k_j(S) \right\}.$$

Using the definition of the coverage vector $\mathbf{y}$ in formula (3.5), the arrival rate of customers at facility $j \in M$ is computed by $\gamma_j = \sum_{i \in N} \lambda_i y_{ij}$. Via formula (3.3), $W(\gamma_j, \mu, k)$ can be calculated. The objective function can be defined with the facility node set M if the facility location vector (3.4) is used.

Hence, the general formulation for this non-linear integer problem is given by (see Aboolian et al., 2008)

$$\min \sum_{j \in M} \left\{ g \sum_{i \in N} \lambda_i d_{ij} y_{ij} + v \sum_{i \in N} \lambda_i y_{ij} W\left(\sum_{i \in N} \lambda_i y_{ij}, \mu, k_j \right) + f_j x_j + h k_j \right\} \tag{3.6}$$

subject to

$$\sum_{j \in M} k_j \leq p, \tag{3.7}$$

$$k_j \leq p x_j, \quad \forall j \in M, \tag{3.8}$$

$$y_{ij} \leq x_j, \quad \forall j \in M, \forall i \in N, \tag{3.9}$$

$$\sum_{j \in M} y_{ij} = 1, \qquad \forall i \in N, \tag{3.10}$$

$$\sum_{k \in M} d_{ik} y_{ik} \leq (d_{ij} - \mathcal{L}) x_j + \mathcal{L}, \quad \forall j \in M, \forall i \in N, \tag{3.11}$$

$$k_j \geq \frac{\sum_{i \in N} \lambda_i y_{ij}}{\mu}, \quad integer, \forall j \in M, \tag{3.12}$$

$$x_j \in \{0,1\}, \quad \forall j \in M,$$

$$y_{ij} \in \{0,1\}, \quad \forall j \in M, \forall i \in N.$$

At most p servers are to be located at all facilities (see Constraint (3.7)). If no service station is installed at node $j \in M$, the Constraints (3.8) and (3.9) forbid assigning servers and demand to this site. Constraints (3.10) ensure that each customer node $i \in N$ is covered by exactly one service station. Let $\mathcal{L}$ in (3.11) be a large positive number (e.g. $\max_{j \in M, i \in N} \{d_{ij}\}$; see Aboolian et al., 2008). By (3.11) it is guaranteed that each customer visits the closest facility. Due to Constraints (3.12), the expected waiting time at a station must be positive.

3.3 Differences in the Models

3.3.1 The Multiple Server Problem

(See Berman and Drezner, 2007.)
p servers are to be located at the facility sites. The objective is not to minimize the total cost of the system, but rather the total cost of all customers (i.e., the sum of the travel related costs and the expected waiting costs at the station). Thus, the only difference to the general model, formulated in the last section, concerns the objective function, which is reduced to

$$\min \sum_{j \in M} \left\{ g \sum_{i \in N} \lambda_i d_{ij} y_{ij} + v \sum_{i \in N} \lambda_i y_{ij} W(\sum_{i \in N} \lambda_i y_{ij}, \mu, k_j) \right\} .$$

3.3.2 The Total Cost Problem

(See Aboolian et al., 2008.)
In this case, the total cost of the system is sought to be minimized (i.e., the fixed costs of installing facilities and the variable costs of servers are also considered). The objective is therefore given by (3.6).

Since variable server costs are considered, the number of servers assigned to the facilities limits itself. It will always be a realistic number and thus no upper bound p is required. Constraints (3.7) and (3.8) can be skipped.

More trade-off relationships between the components of the objective are implicated by the objective function of the Total Cost Problem. Not only is there a trade-off between the number of servers and the expected waiting time (see Section 3.1). Locating fewer facilities on the network diminishes the costs of installing server sites and leads to pooling of servers. Thus, customers' waiting times at the stations are reduced as well. On the other hand, assigning fewer server units to the facilities reduces the variable server costs. However, the waiting times of the customers are increased.

For both problems, the difficulty is now to find the optimal number of facilities, the optimal location of these sites, and the optimal assignment of servers for each service facility.

3.4 The Optimal Assignment of Servers

The set $S \subset M$ of nodes, where facilities are placed, is supposed to be given. The optimal assignment of servers $\mathbf{K}^*(S) = \left\{k_j^*(S) \,|j \in S\right\}$ for the given S is sought. Since the travel related costs and the fixed costs of installing servers at a service site do not depend on the number of assigned servers $k_j^*(S)$, the objective function is given by

$$\min_{\left\{k:k_j \geq k_j^{min}\right\}} \left\{ F(\mathbf{K}) = \sum_{j \in S} \left[hk_j + v\lambda_j(S) W(\lambda_j(S), \mu, k_j)\right] \right\}.$$

In order to meet all customer demands, the ratio between the arrival rate $\lambda_j(S)$ and the total service rate of all servers $k_j\mu$ must be less or equal to one. The optimal number of servers k_j^* assigned to each facility $j \in S$ must thus be at least k_j^{min}. This lower bound can be calculated by

$$k_j^{min} = int \left[\frac{\lambda_j(S)}{\mu}\right] + 1,$$

where $int\,[.]$ rounds the parameter down to the next integer. $k_j^{min}(S)$ represents the minimum number of servers required to meet the congestion at each node $j \in S$.

The following greedy algorithm[1] finds the optimal assignment vector $\mathbf{K}(S)$ for a given set of facility nodes S and a given upper server limit p.

Algorithm 3.1: Assignment of Servers for the Multiple Server Problem

```
For each facility $j \in S$ follow these steps:
*Step 1*: Find the set of customer nodes $C_j(S)$ that are assigned to
          site $j$ and compute the demand rate $\lambda_j(S) = \sum_{i \in C_j(S)} \lambda_i$.
*Step 2*: Calculate the minimum number of servers required to serve
          the demand $\lambda_j(S)$ by $k_j^{min} = int\left[\frac{\lambda_j(S)}{\mu}\right] + 1$ and set $k_j =$
          $k_j^{min}$.
*Step 3*: If $\sum_{j \in S} k_j > p$, the customer demand can never be met
          and thus there is no feasible solution. Else go to *Step 4*.
*Step 4*: $l = \arg\ \max_{\{j \in S\}} \{W(\lambda_j(S), \mu, k_j) - W(\lambda_j(S), \mu, k_j + 1)\}$
          and set $k_l = k_l + 1$.
*Step 5*: If $\sum_{j \in S} k_j < p$ go to *Step 4*, else stop.
```

[1]A greedy algorithm is a shortsighted one. At each step the local optimal solution is sought, hoping that this leads to the desired global optimum.

An optimal server assignment $\mathbf{K}(S) = \left\{k_j^*(S)\,|j \in S\right\}$ is found.

Algorithm 3.1 can be used for the Multiple Server Problem (see Berman and Drezner, 2007). In the case of the Total Cost Problem, there is no upper limit on the number of servers. Thus, the algorithm must be changed (see Aboolian et al., 2008).

The contribution of each service node $j \in S$ to the objective function value is defined by

$$F_j(k_j) = hk_j + v\lambda_j(S)W(\lambda_j(S), \mu, k_j).$$

Thus, $F(\mathbf{K}) = \sum_{j\in S} F_j(k_j)$. The optimal assignment of servers can be found by checking the decrease in the objective function value. Only the last three steps of Algorithm 3.1 must be changed.

Algorithm 3.2: Assignment of Servers for the Total Cost Problem

```
For each facility $j \in S$ follow these steps:
Step 1:  Find the set of customer nodes $C_j(S)$ that are assigned to
         site $j$ and compute the demand rate $\lambda_j(S) = \sum_{i\in C_j(S)} \lambda_i$.
Step 2:  Calculate the minimum number of servers required to serve
         the demand $\lambda_j(S)$ by $k_j^{min} = int\left[\frac{\lambda_j(S)}{\mu}\right] + 1$ and set $k_j = k_j^{min}$.
Step 3:  If $F_j(k_j+1) - F_j(k_j) \geq 0$, go to Step 5. Otherwise continue
         with Step 4.
Step 4:  Set $k_j = k_j + 1$ and go to Step 3.
Step 5:  Set the optimal number of servers for facility $j$, $k_j^*(S) = k_j$,
         and $F_j(k_j^*(S)) = hk_j^*(S) + v\lambda_j(S)W(\lambda_j(S), \mu, k_j^*(S))$. Stop.
```

Again, an optimal server assignment for all nodes $j \in S$ is found.

Property 3: $F_j(k_j) = hk_j + v\lambda_j(S)W(\lambda_j(S), \mu, k_j)$ is a sum of linear and convex functions and hence also convex in k_j.

This property follows from the convexity of $W_q(\lambda_j(S), \mu, k_j)$ (see Property 2). It guarantees that the solutions found by Algorithm 3.1 and Algorithm 3.2 are optimal assignments of servers for a given set S.

Chapter 4

Metaheuristics for Server Location Problems

The algorithms formulated in the previous chapter (see Section 3.4) find the best allocation $\mathbf{K}(S) = \left\{k_j^*(S) : j \in S\right\}$ of server units to a given set of facility nodes S. The problem is now to find this optimal set S of facility sites, which minimizes the objective function. Algorithm 3.1 or Algorithm 3.2 can then be used to calculate an assignment $\mathbf{K}$.

The objective function depends on the number of assigned servers $\mathbf{K}$ and on the service nodes S. It can thus be written as

$$F(\mathbf{K}, S) = \sum_{j \in S} \left\{ g\lambda_j(S)d_{ij} + v\lambda_j(S)W_q(\lambda_j(S), \mu, k_j) + f_j + hk_j(S) \right\}.$$

For a given S we define $F(S) = \min_{\mathbf{K}} \{F(\mathbf{K}, S)\}$.

For the Total Cost Problem, Aboolian et al. (2008) applied an exact algorithm. However, this procedure performs well only for medium size problems.

In the following we concentrate on metaheuristics. Glover and Kochenberger (2003) define these searching procedures as follows:

> Metaheuristics [...] are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima [...].

Several heuristics (Descent Algorithm, Simulated Annealing, Genetic Algorithm, and Tabu Search) are introduced in order to solve our formulated problems and to obtain a set S of service locations that minimizes $F(S)$.

For all of the algorithms introduced in this chapter a neighborhood of the given set S is needed. Therefore, the neighborhood of a set has to be defined.

4.1 Neighborhoods

First of all, the definition of neighborhoods used here is totally different from the one in topology[1]. In the underlying consideration there is no topological space, and a neighbor does not have to contain all nodes of the original set.

Giving a common definition of a neighborhood is not possible. There is a huge number of ways, how to select neighbors of a given solution. For each problem it is different. Therefore, an universal way of defining neighborhood does not exist. However, it can be said that a neighborhood of a given solution set S is given by a set that can be achieved by a simple local movement.

For our facility location problems the following neighborhood that can be obtained in three different ways is used (see, e.g., Aboolian et al., 2008):

1. Subsets with one additional node to S. There are $n - |S|$ elements in the complement and thus $n - |S|$ sets have an additional node to S.
2. Subsets obtained by removing one node from S. It is obvious that $|S|$ new sets arise.
3. Subsets with one additional node to S and one node removed from S. In this case, an element of S is replaced by an element of the complementary set $n - |S|$. Therefore, $|S|\,(n - |S|)$ new neighbors are found.

Consequently, the neighborhood of a node set S features a cardinality of $n + |S|\,(n - |S|)$. The cardinality of the solution set S is not fixed (depends on the current solution) and n denotes the total number of customer and facility nodes in the considered network.

[1]In topology a set N is called neighborhood of a point p in a topological space if N contains an open set O such that $p \in O \subseteq N$. Furthermore, the set N is called neighborhood of set G if an open set O exists such that $G \subseteq O \subseteq N$ (see, e.g., Heuser, 2004).

4.2 Classification of Search Techniques

In the following, the expression "NP-hard" problem is used several times. Therefore, it is explained at this point (see Heaton, 2008).

Today, many problems are solved by computers in polynomial time and thus are not NP-hard. If the number of steps to solve a problem is bounded by a polynomial, it is referred to as P-problem (polynomial problem). Problems that cannot be solved in polynomial time are called NP-hard (nondeterministic polynomial hard problems). The possible solutions of such problems often increase at a much greater rate than exponentially. The increasing rate of NP-hard problems is described by the factorial operator $n!$.

As shown in Figure 4.1, the search techniques can be divided into three classes: calculus-based, random, and enumerative techniques (see Ribeiro Filho et al., 1994).

The first, calculus-based (numerical) techniques require certain necessary conditions that have to be fulfilled by the optimal solution. By setting the gradient of the objective function to zero and solving these equations, the indirect calculus-based methods find local extrema. On the other hand, the direct methods only assess the gradient in order to find the right direction. Newton's or Fibonacci's method exemplify these kind of methods.

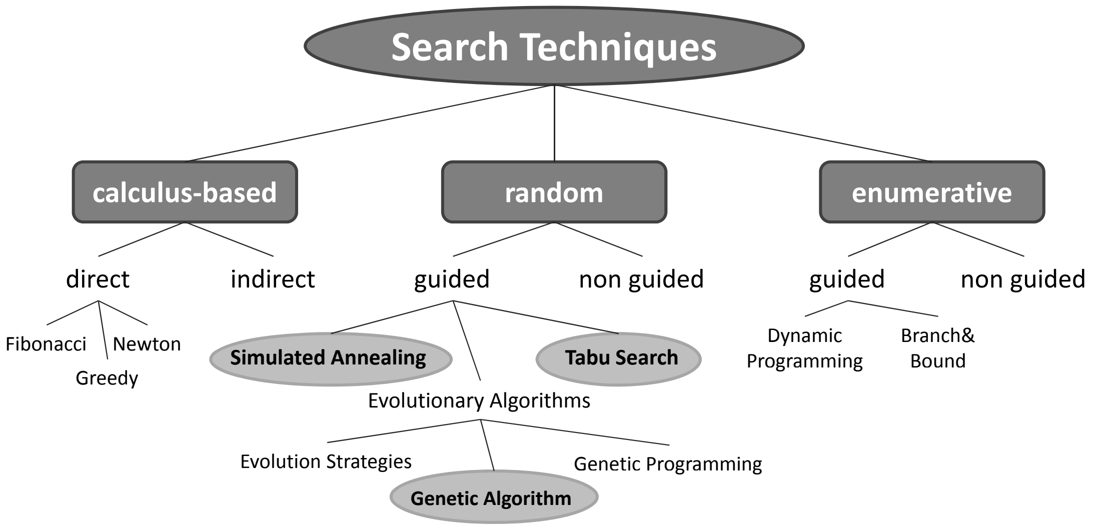

Figure 4.1: Classes of Search Techniques

In contrast to numerical techniques, enumerative techniques provide an exact solution rather than an approximate one. Implementing enumerative techniques is very simple. However, the domain space of many problems is too large since enumerative techniques consider each point. Dynamic Programming exemplifies this technique.

These two methods are not suitable for the underlying models. As mentioned above, the numerical techniques require certain necessary conditions. These prerequisites are not fulfilled for our problems and thus the calculus-based techniques cannot be applied. Since our models are NP-hard, the number of possible solutions is very high. Thus, an enumerative algorithm would take very long to compute all potential solutions.

Due to these facts we will concentrate on the third class, the random techniques. These so-called heuristics only check a subset of all possible solutions using some strategies. Heuristics need much less time to execute, however, in most of the cases the optimal solution will not be found. It is also unknown, how far the resulting solution differs from the best.

Random search techniques are appropriate for location problems, because they solve very complex problems. Being based on enumerative techniques, the random search class also uses additional information to find the optimal solution. While Simulated Annealing is based on a thermodynamical process, Evolutionary Algorithms simulate natural selection procedures and Tabu Search uses memory.

4.3 Descent Procedure

An iterative procedure goes from a current feasible solution to another one. This new solution is either accepted or the algorithm returns to the last accepted solution. If this next solution is from the neighborhood of the current one, the iterative method is called neighborhood search algorithm. A popular example for these searching procedures is the Descent Algorithm.

Both Berman and Drezner (2007) and Aboolian et al. (2008) develop a Descent Procedure to solve the formulated problem. With the help of the neighborhood definition from Section 4.1, this algorithm is simple and straightforward.

In a nutshell, it can be described as follows: Starting with a random, initial set S, the values of the objective function for all neighbors of S are calculated. If the minimum of all these values is smaller than the objective function value of the starting S, the procedure is repeated with the new minimal S. Otherwise, it stops and returns to the last accepted S.

Algorithm 4.1: Descent Procedure

```
Step 1: Randomly generate a starting $S$ with $|S| = 1$ and calculate
        $F(S)$ (Algorithm 3.1 or 3.2).
Step 2: For all subsets $S'$ in the neighborhood of $S$ calculate $F(S')$.
        Set $F_{min} = \min\{F(S')\}$ and $S_{min} = \arg\min\{F(S')\}$.
Step 3: If $F_{min} < F(S)$, set $S = S_{min}$ and $F(S) = F_{min}$. Go to
        Step 2.
Step 4: Otherwise, stop. The best solution found by the Descent
        Procedure is $S$.
```

As it is pointed out in Aboolian et al. (2009), this algorithm finds a feasible solution. Unfortunately, it is not optimal most of the time. Disconnected "islands" of feasible solutions generate the solution space. If the Descent Procedure reaches such an island, it cannot move to another one. Therefore, only the best solution on this island can be found (i.e., the method may stop at an extremum that is locally optimal but not a global optimum.).

In the following sections, algorithms that deal with this kind of problem are presented.

4.4 Simulated Annealing

Simulated Annealing is a generic Monte-Carlo Algorithm[2] for combinatorial optimization problems. It is often used for large scale problems that belong to the class of NP-complete (nondeterministic polynomial time complete) problems.

The method generates several sample states (permutations) of a statistical thermodynamic system. It was inspired by the annealing of metallurgy. If melted metals are cooled controlled, the defects in the solid state body can be reduced and the resulting crystals become bigger. This thermal process is simulated in order to find a global extremum of optimization problems.

In the mid 1980s, Kirkpatrick et al. (1983) and Černý (1985) introduced this Simulated Annealing method. Originally it was developed for finding the optimal design of a computer (see Kirkpatrick et al., 1983). Another very popular application of the Simulated Annealing Algorithm is the traveling salesman problem (see, e.g., Černý, 1985).

4.4.1 Basic Physical Background

In order to interpret and understand the process of the Simulated Annealing Algorithm, the physical background is of great importance. It is very simple and well established.

As already mentioned above, the Simulated Annealing Algorithm is based on the annealing of solids (e.g. metals). This thermal process can be described in the following six steps:

1. The solid is heated in a head bath, where the temperature is increased until a certain upper bound is reached.
2. The solid is melting and its atoms have high-energy values due to the temperature in the head bath. The particles are randomly arranged and free to restructure themselves.
3. Slowly the temperature of the head bath is lowered again and the energy of the atoms decreases.
4. A thermal equilibrium (steady state) is obtained at each temperature level.

[2] A Monte-Carlo Algorithm is a stochastic method to obtain numerical solutions. The repeated random sampling might produce an incorrect result with a certain bounded probability. But compared to deterministic algorithms, this method is often more efficient.

5. The atoms can arrange themselves in an energetic optimal structure, because the temperature level is kept around the freezing point for a long time.
6. If no more changes occur, the procedure is terminated. The minimum energy state of the system is achieved.

The most important requirement in this procedure is that the annealing must be carried out really slowly. Since a very stable structure of the resulting crystal is desired, the temperature of the head bath should be lowered very carefully. No equilibrium is achieved if the cooling process is too fast. As a consequence, many defects and irregularities in the structure occur.

A cooling procedure that is carried out too fast might cause defects. This can be compared with a local extremum of an optimization problem. If the temperature is lowered too fast, the algorithm might end up in an local optimum.

4.4.2 The Algorithm

To simulate this thermal process, a Monte-Carlo method suggested by Metropolis et al. (1953) is used. The authors develop a simple algorithm that generates iterative improvements of states. However, even worse solutions than the current state are accepted with a certain probability.

The Metropolis Algorithm can be described as follows (see Kirkpatrick et al., 1983):

We start from a certain state i with energy E_i. In each step of the algorithm a change is made to obtain a new state j with energy E_j. Typically, only small changes are made. The difference in the energy is computed in the next step and denoted by $\Delta E = E_j - E_i$.

If $\Delta E \leq 0$, the new state j has lower energy and is therefore accepted. It is now used as a starting state for the next iteration.

On the other hand, if $\Delta E > 0$, a random, uniformly distributed number between 0 and 1 is generated. If $P(\Delta E) = \exp(\frac{-\Delta E}{k_B T})$ (k_B denotes the Boltzmann constant and T the temperature) is bigger than this random number, the new state j is accepted. Otherwise, the initial state i is used for the next iteration.

These steps are repeated either until T equals zero or until a given upper bound of iterations is reached.

This algorithm by Metropolis et al. (1953) simulates the configuration of the atoms within the head bath. As we can see, moves in the "wrong" direction (i.e., states with higher energy) are also accepted when the temperature T is high enough. As T is lowered, $P(\Delta E)$ decreases. If T goes to zero, $P(\Delta E)$ converges towards zero and therefore only states with lower energy are accepted.

The fact that even movements with higher energy are allowed with a certain probability decreases the chance of ending up in a local minimum.

For the sake of completeness, also the pseudo code of the Simulated Annealing Algorithm is formulated:

```
begin
  t = T(0); n = 0;                  // starting temperature; iteration;
  s = s0; e = E(s);                 // initial state; initial energy;
  sbest = s; ebest = e;             // current best solution;
  repeat
    choose s* in N(s);              // select a neighbor of s;
    snew = s*; enew = E(s*);        // calculate its energy;
    d = enew-ebest;                 // difference in energy;
    if (d < 0)                      // energy improvement?
      sbest = snew; ebest = enew;
    else if (exp(-d/(t*k_b)) > rand(0,1))
    s = snew; e = enew;
    t = T(n); n = n+1;              // current temperature; iterations;
  until n >= nmax;                  // nmax = number of iterations;
  return sbest                      // return best found solution;
end;
```

In order to adapt this algorithm to a general minimization problem, the cost function (objective of the problem) is used instead of the energy. And in place of the configurations (states of the system) a set of parameters of the decision variables is used. Proceeding like this, we obtain the Simulated Annealing Algorithm, where the temperature T denotes a control parameter.

Summing up, the Simulated Annealing Algorithm solves minimization problems with a high number of variables by feigning the cooling process of melted metal. In each step a random solution in the neighborhood is considered. Depending on the temperature T and the difference of the function values, a probability decides about the next state.

4.4.3 Formulation of the Simulated Annealing Algorithm

For the problems described in Section 3.2 (the Multiple Server Problem and the Total Cost Problem) the Simulated Annealing Algorithm is formulated as follows (see Aboolian et al., 2008; Berman and Drezner, 2007):

Algorithm 4.2: Simulated Annealing

```
Step 1: Set temperature $T$ to the starting temperature $T_0$ and the
        number of iterations $i$ to 1.
        Generate a set of facility sites $S$ randomly and calculate
        $F(S)$.
Step 2: Define $S'$, a random move in the neighborhood of $S$.
Step 3: Calculate $F(S')$.
Step 4: If $F(S') < F(S)$, set $S = S'$ and $S^{best} = S'$. Continue with
        Step 6.
Step 5: Otherwise, calculate $\delta = \frac{F(S')-F(S)}{T}$ and generate a random
        number $rand$ between 0 and 1. If $e^{-\delta} > rand$ change $S$ to
        $S'$.
Step 6: Set $T = \alpha T$ and $i = i + 1$. If $i < iter$, go to Step 2.
Step 7: The last value of $S^{best}$ is the best solution found by this
        algorithm.
```

As we can see, three parameters have to be determined before starting the procedure. First of all, the starting temperature T_0 of the head bath has to be defined. In each iteration the current temperature is reduced by a certain factor $\alpha < 1$. This factor has to be fixed before starting the algorithm. And finally, the maximal number of iterations *iter* has to be predetermined.

4.5 Genetic Algorithm

Both Simulated Annealing and Genetic Algorithms belong to the group of guided random search techniques. While Simulated Annealing is a thermal process with a physical background, Genetic Algorithms are combinatorial evolutionary procedures based on the mechanics of biological evolution. Alp et al. (2003) describe Genetic Algorithms in the following way:

> Genetic Algorithms (GAs) are heuristic search methods that are designed to mimic the evolution process. New solutions are produced from old solutions in ways that are reminiscent of the interaction of genes. GAs have been applied with success to problems with very complex objective functions. While a number of applications to combinatorial optimization problems have been reported in the literature, there are few applications of genetic algorithms to facility location problems.

John Holland was the first to develop Genetic Algorithms. In 1975 he published the book "Adaptation in natural and artificial systems" (Holland, 1975), which presents the theory of Genetic Algorithms. Further very important contributions concerning these optimization techniques were delivered by David Goldberg, a student of Holland (Goldberg, 1989). Dawid (1996) provides analytical results and applications to economic models using adaptive learning by Genetic Algorithms.

4.5.1 Evolution and the Genetic Algorithm

Charles Darwin's theory of evolution (see Darwin, 1859) can be seen as the initial point for Genetic Algorithms. He found out that organisms best suited to their environment have a higher probability to survive and to produce offspring. The parents pass their adaptations on to the children and the new generation is then even better adjusted. Thus, life and its quality is improving over time. This theory became known as "Survival of the fittest".

At the beginning, a starting population (organisms) exists. Each member can be characterized by their genetic information (represented by the chromosomes). Due to their fitness (suitability to the environment), they have a certain survival probability. Surviving organisms produce a further generation, whose chromosomes originate from manipulating the genetic information of their parents. Either crossover or mutation is used as genetic operator (for details see Ribeiro Filho et al., 1994). The new generation serves as new starting population and the circle is repeated.

As we can see, evolution demonstrates how to solve the problem of producing organisms that can survive in a particular environment. This biological evolution is now simulated to formulate the algorithm.

4.5.2 The Algorithm

The starting population in a Genetic Algorithm corresponds to a set of possible solutions of the underlying problem. This set has to be generated randomly at the beginning of the procedure. The solutions (chromosomes) are evaluated by their value of the objective function (fitness function). Based on these values, the solutions are selected to serve as parents. Pairs of the chosen solutions are now crossed to produce an offspring. The steps of the algorithm can be illustrated in a simple figure (see Figure 4.2).

Summing up, the algorithm consists of the following four components that are also shown in the figure:

1. a population (set) of possible solutions;
2. an objective (fitness) function;
3. a selection operator for the parents; and
4. a genetic operator to produce a new generation.

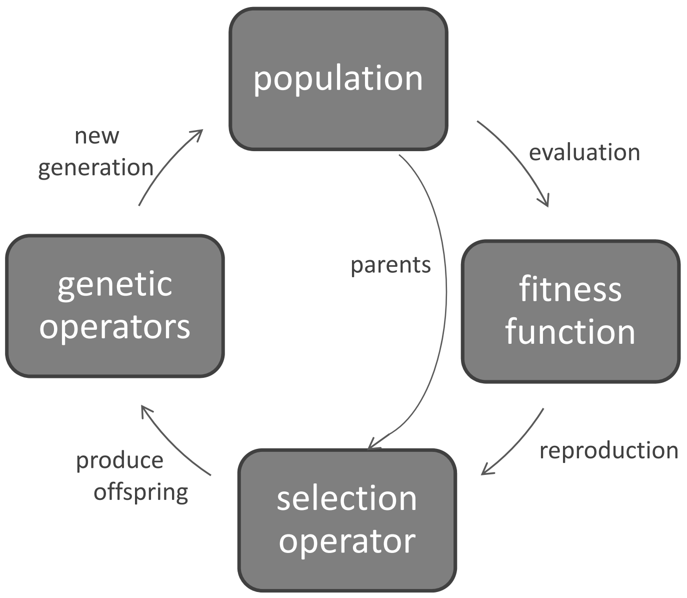

Figure 4.2: The "Reproduction" Circle of the GA

4.5.3 Formulation of the Genetic Algorithm

The Genetic Algorithm is formulated for the problems in Section 3.2. A population consists of P members (solutions) and G new generations have to be created until the algorithm terminates. An offspring is produced following a certain merging procedure. The new member is admitted to the population if a worst solution already exists (see Berman and Drezner, 2007).

Algorithm 4.3: Genetic Algorithm

```
Step 1: Randomly generate $P$ solutions for the starting population
        and perform the Descent Procedure (Algorithm 4.1) on all
        these solutions.
Step 2: Repeat the following steps $G$ times:
        • Select two random members. By using the merging pro-
        cedure create a new population member $S'$.
        • If $F(S')$ overvalues one of the population members:
          - If $S'$ is identical to one of the population members, do
        not change anything. Start the next iteration.
          - Otherwise, instead of the worst member admit $S'$.
        • If $F(S')$ is worse than all the other population members,
        start the next iteration.
Step 3: The best member of the population denotes the solution
        found by the algorithm.
```

Berman and Drezner (2007) and Aboolian et al. (2009) use the following merging procedure in order to create a new offspring. The node sets S_1 and S_2 denote the parents.

1. Create $S^U = S_1 \cup S_2$ (union of the parents).
2. Create $S^I = S_1 \cap S_2$ (intersection between the parents).
3. Create S^D in the following manner: to all nodes that are in S^U but not in S^I add three random nodes outside of S^U.
4. Create S' (starting offspring) by adding a random node from S^D to S^I.
5. Apply a restricted Descent Algorithm to S' by removing or adding nodes only in S^D. The nodes in S^I are fixed.
6. The new produced offspring is denoted by the solution of the procedure.

4.6 Tabu Search

The last algorithm introduced in order to solve the underlying optimization problems is the Tabu Search. Besides Simulated Annealing, this method is also an iterative search technique based on the definition of neighborhood. However, Tabu Search is not adapted from a thermodynamical process. The most essential feature of the method described in this section is its "use of memory".

Fred Glover was the first to concentrate on memory mechanism. In 1986 he introduced Tabu Search (see Glover, 1986). A full description of the procedure is given in Glover (1989).

Tabu Search is applied in a large diversity of problems. Glover and Laguna (1997) mention in their book among others production planning and scheduling, routing telecommunications, resource allocation, transportation and network design.

4.6.1 Use of Memory

As already mentioned, the most essential component of Tabu Search is its use of memory. Due to this memory, the method will avoid local minima and is prevented from cycling.

We will only concentrate on the most important part, the short term memory. Additionally, the intermediate and the long term memory intensify and diversify the search.

Every step of the Tabu Search is looking for the best solution in the neighborhood. This searching strategy might lead to cycles if no memory is used. The short term memory prevents the algorithm from visiting a solution twice and thus cycles are avoided. Since certain moves are taboo, this searching procedure is called Tabu Search.

In Tabu Search the used memory can be both explicit and attributive. An explicit memory consists of complete solutions that have already been visited during the search. These solutions in the memory are used to guide the searching procedure and to avoid visiting solutions more than once. Contrary to this, information about solution attributes is recorded in the attributive memory. In a network, for example, arcs from one node to another are memorized. This also guides the process in the right direction.

4.6.2 The Algorithm

Tabu Search (TS) starts with a feasible initial solution (obtained by the Descent Algorithm). The next step is to create candidates in the neighborhood of the current solution. If the best neighborhood candidate provides an improvement and is not yet included in the tabu list, it is accepted as new solution. In order to avoid visiting this solution again, the tabu list is updated. Otherwise, if the best neighborhood candidate does not provide an improvement, the current solution remains the same and only the tabu list is updated. This procedure is repeated iteratively until a stopping criterion is met.Tabu Search might stop if:

- a certain number of iterations is reached;
- the solution has not improved for a while;
- the neighborhood is empty; or
- a satisfying solution has been found.

The procedure of a Tabu Search is illustrated in Figure 4.3.

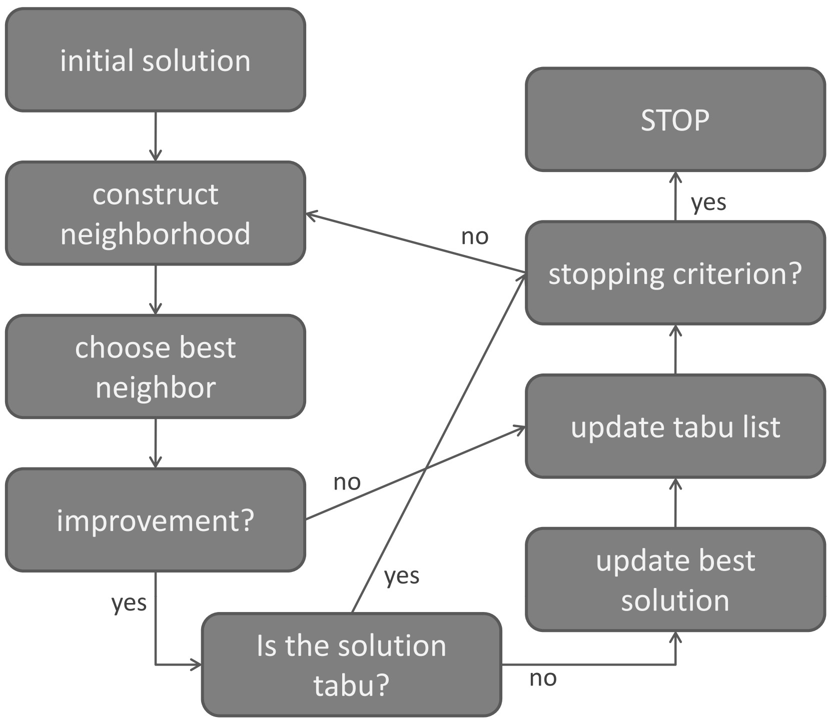

Figure 4.3: Procedure of the TS

4.6.3 Formulation of the Tabu Search

Again the algorithm for the underlying problems is formulated. Thereby the Descent Procedure from Section 4.3 and the definition of a neighborhood from Section 4.1 are used.

Algorithm 4.4: Tabu Search

```
Step 1: Use the Descent Algorithm to generate a starting solution $S$. Set $F^* = F(S)$ and $S^* = S$.
Step 2: Empty the tabu list.
Step 3: For each subset $S'$ in the neighborhood of $S$ calculate $F(S')$.
Step 4: Set $F_{min} = \min\{F(S') : S' \text{ is a neighbor of } S\}$ and set $S_{min} = \arg\min\{F(S') : S' \text{ is a neighbor of } S\}$.
Step 5: If $F_{min} < F^*$ and if $S_{min}$ is not in the tabu list, change the current solution $S$ and $S^*$ to $S_{min}$. If the stopping criterion is met, stop and return $S^*$. Otherwise, go to Step 2.
Step 6: Otherwise:
        • Set $S' = \arg\min\{F(S') : \text{move is not in the tabu list}\}$ and use $S'$ for the next iteration ($S = S'$).
        • The move from $S$ to the selected $S'$ is added to the tabu list.
        • The nodes, whose tenure exceeds the one of the tabu list, are deleted from the list.
        • Go to Step 7.
Step 7: If the stopping criterion is not satisfied, go to Step 3. Otherwise, stop with the resulting solution $S^*$.
```

After *Step 5*, when a better solution is found, the tabu list is emptied each time. This leads to a more flexible searching process.
In *Step 6* the moves being not in the tabu list are nodes that have been in S at the beginning but are not in the selected S' or nodes being in S' but not in the starting S.

Chapter 5

Computational Results and Sensitivity Analysis

John Beasley provides and maintains an OR-Library, a collection of test problems for a number of different areas of Operations Research. Originally, the OR-Library was described in Beasley (1990) and the test data sets were distributed by electronic mail. Now they are also available online[1].

In this thesis, the 40 problems from Beasley (1985) suggested for the uncapacitated p-median problem are selected for the computational experiments. The objective of a p-median model is to locate p facilities on a network in order to minimize the distance between all customers and their closest facility. Since there is no upper bound on the number of facilities in the underlying models, p is used to be the maximal number of servers assigned to all service sites. (This value is only used in the Multiple Server Problem. Considering the Total Cost Problem, no upper bound on the number of servers is given. Thus, the value of p is simply neglected.)

The problems suggested by Beasley range from 100 to 900 demand points and from 5 to 200 facilities (server units). It goes without saying that problems with a high number of nodes and a high number of servers require more running time. Therefore, only 10 problems (1, 2, 6, 7, 11, 12, 16, 17, 21, and 22) are chosen, for which the examinations in this thesis are carried out.

All programs are coded in *C#* and run on a computer with a dual core processor with 2,4 GHz and 4048 MB RAM. An example of the implemented algorithms is given in Appendix A.3.

[1]see http://people.brunel.ac.uk/∼mastjjb/jeb/info.html

5.1 Computational Experiments

5.1.1 Determination of Parameter Values

First of all, suitable values for the parameters that appear in the model formulations have to be found. When choosing these values, the suggestions made in Berman and Drezner (2007) are strongly followed. The parameters used in their implementations are a result of extensive experiments with each of them.

The demand rate λ_i at customer node i is set to 1. To simplify matters λ is assumed to be equal for all nodes.

The parameter θ is defined to be the ratio between total service capacity and total demand of the customers ($\theta = \frac{p\mu}{n\lambda}$). It is presumed to be 1.1. Thus, the service capacity of all servers is 10% more than the total demand, which leads to normal service conditions. Consequently, the service rate μ equals $\theta\frac{n\lambda}{p}$ for all server units at all facility stations.

The fixed costs of installing servers f_j at facility node j are also assumed to be equal for all nodes and are set to 1000. h, the variable server costs, are presumed to be 50 for all servers. The customer's travel costs g per unit of time and the customer's waiting costs v per unit of time are set to 1.

In Table 5.1 the above mentioned assumed parameter values for the heuristics are summarized.

parameter	value	additional information
λ_i	1	$\forall$ customer nodes i
θ	1.1	$\theta = \frac{p\mu}{n\lambda}$
μ	$\theta\frac{n\lambda}{p}$	$\forall$ servers
f_j	1000	$\forall$ facilities j
h	50	-
g	1	-
v	1	-

Table 5.1: Determination of Parameter Values

5.1.2 Metaheuristic Parameters

Before starting the algorithms, one also has to decide how often the four procedures should be run for each problem and how many iterations the three heuristics should carry out. In addition, the starting temperature and the reducing factor must be assigned for the Simulated Annealing Algorithm.

The running times of the Descent Procedure are on average under one second. Thus, it is run 500 times for each problem.

For the Simulated Annealing Algorithm, $N = 4000n\sqrt{p}$ iterations are carried out and the number of runs equals 50 for each problem. The initial temperature T_0 is set to 1000 and it is reduced by the factor $\alpha = 1 - \frac{5}{N}$ after each iteration.

The Genetic Algorithm is executed for 500n generations with a population size of n. The running times of this algorithm are relatively high, compared to the one of the Descent Procedure. In case of the Genetic Algorithm, one run for the ten chosen problems takes on average 12 minutes. Therefore, each problem is run only 50 times.

The running time of the Tabu Search Algorithm is lower. However, it is also carried out only 50 times, and $5n$ iterations are made.

Table 5.2 gives an overview about all these metaheuristic parameters.

algorithm	# runs	# iterations	parameters
Descent Procedure	500	-	-
Simulated Annealing	50	$N = 4000n\sqrt{p}$	$T_0 = 1000$, $\alpha = 1 - \frac{5}{N}$
Genetic Algorithm	50	$500n$	population size $= n$
Tabu Search	50	$5n$	-

Table 5.2: Metaheuristic Parameters

These parameters are used for the Multiple Server Problem. For the second model, the Total Cost Problem, the number of iterations and the parameters remain the same. Only the number of runs is changed as follows:

The Descent Procedure is again executed 500 times. All the other heuristics are run only 30 times.

5.1.3 The Multiple Server Problem

In Table 5.3 the best known results (i.e., the minimal cost value) obtained by the four procedures in our computational experiments are presented.

As usual, the number of nodes in the network and the maximal number of servers placed at the facilities are denoted by n and p. The objective function value (the costs) of the best found solution is named minimal costs. q is the number of server nodes for the best known solution and max the maximal number of server units assigned to one facility site. The abbreviations for the procedures are straightforward: Descent for Descent Procedure, GA for Genetic Algorithm, SA for Simulated Annealing, and TS for Tabu Search.

n	p	minimal costs	q	max	found by
100	5	6692.49	4	2	TS and GA
100	10	5309.07	7	3	TS and GA
200	5	8172.77	4	2	GA
200	10	6709.94	8	3	GA
300	5	8265.45	5	1	TS
300	10	7577.84	9	2	TS
400	5	8207.07	5	1	TS
400	10	7609.77	9	2	GA
500	5	9520.79	5	1	SA
500	10	9415.10	9	2	TS

Table 5.3: Best Known Results for the Multiple Server Problem

As one can see, in the majority of cases, the best known result is found by the Tabu Search Algorithm (six times). The Genetic Algorithm detected the best known result for five problems.

For one problem, the Simulated Annealing Algorithm found the best result. Due to the fact that this algorithm is very much based on coincidence and detected only once the best solution, we do not pay much attention to this result.

The Descent Procedure never obtained a best solution. Its found objective function values are higher than the best known solution for all problems.

Can it be inferred from this result that the Tabu Search is the best algorithm to solve the Multiple Server Problem? Does the Genetic Algorithm provide adequate solutions as well?

In order to answer these questions, a detailed look at the results of all procedures is presented. In particular, their deviation from the best found result is compared.

Results for the Four Heuristics

In the following tables (see Table 5.4 and Table 5.5), the deviations from the best known objective function values for the Descent Procedure, the Simulated Annealing Algorithm, the Genetic Algorithm, and the Tabu Search are presented (i.e., $\frac{\text{result from the particular procedure}}{\text{best known result}}$).

min denotes the minimal, *ave* the average, and *max* the maximal deviation from the best found cost value. All these values are expressed as a percentage over the best known (BK) result from Table 5.3.

		Descent			Simulated Annealing		
		% over BK result			% over BK result		
n	*p*	*min*	*ave*	*max*	*min*	*ave*	*max*
100	5	2.79	7.70	16.88	0.13	8.35	14.19
100	10	10.41	16.07	21.41	8.87	17.19	24.51
200	5	0.50	9.84	18.78	0.71	9.12	15.41
200	10	1.39	8.57	13.61	6.45	12.11	22.62
300	5	1.05	6.79	8.07	0.93	6.04	10.16
300	10	2.44	9.50	19.38	10.58	16.88	21.57
400	5	0.26	9.47	15.77	1.29	8.49	15.03
400	10	4.67	11.47	19.98	9.17	17.41	31.73
500	5	6.68	12.83	17.30	0.00	7.58	14.86
500	10	1.31	9.20	15.34	3.97	10.95	15.00
Average		3.15	10.14	16.65	4.21	11.41	18.51

Table 5.4: Descent- and Simulated Annealing- Results for the Multiple Server Problem

The solutions found by the Genetic Algorithm are on average only 1.90% higher than the best found objective function value. Since this procedure also detected the best known solution in 50% of all problems, it turns out to be the most efficient one of all algorithms.

		Genetic Algorithm			Tabu Search		
		% over BK result			% over BK result		
n	*p*	*min*	*ave*	*max*	*min*	*ave*	*max*
100	5	0.00	2.89	4.14	0.00	4.86	10.24
100	10	0.00	2.91	3.94	0.00	9.39	17.13
200	5	0.00	0.12	0.50	0.02	1.28	6.97
200	10	0.00	1.26	2.11	1.78	4.31	9.51
300	5	0.01	0.01	0.04	0.00	4.80	6.21
300	10	2.44	2.50	3.96	0.00	8.86	19.38
400	5	0.26	0.30	2.02	0.00	7.64	10.67
400	10	0.00	3.51	5.42	3.98	8.31	16.87
500	5	2.90	3.93	6.68	6.49	10.19	12.16
500	10	1.31	1.61	3.04	0.00	6.54	9.77
Average		0.69	1.90	3.19	1.23	6.61	11.89

Table 5.5: Genetic Algorithm- and Tabu Search- Results for the Multiple Server Problem

The second best procedure, considering these tables, is the Tabu Search Algorithm. Although it found the best solution in most of the cases, its results are on average already 6.61% higher than the best found solution.

The other two procedures did not perform in a satisfying way (see Table 5.4). On average, their results are over 10% higher than the best known solution.

Conclusions for the Multiple Server Problem

Concerning the objective function value, the Tabu Search Algorithm detected the best results in the majority of cases. However, taking not only the best known solution, but all the found results into consideration shows that the Genetic Algorithm performed really well and is therefore recommended to use for the Multiple Server Problem. The Simulated Annealing and the Descent Procedure did not meet the expectations, because their results exceed the optimal solution clearly.

5.1.4 The Total Cost Problem

For the Total Cost Problem, the best known solutions obtained by the four heuristics in our computational experiments are presented in Table 5.6.

Again, n denotes the number of nodes and p the number of servers placed at the facilities. As before, q is the number of server nodes for the particular solution and max the maximal number of server units assigned to one facility.

n	minimal costs	p	q	max	found by
100	10254.36	6	2	4	all 4 procedures
100	10301.75	10	2	5	all 4 procedures
200	12038.34	6	2	4	TS, GA and Descent
200	11350.05	10	1	10	all 4 procedures
300	1150.54	5	2	3	TS, GA and Descent
300	13024.96	10	3	4	GA and Descent
400	12146.60	5	2	3	TS, GA and Descent
400	13216.07	10	2	6	TS, GA and Descent
500	13625.17	6	3	2	TS and GA
500	15049.79	10	3	4	TS, GA and Descent

Table 5.6: Best Known Results for the Total Cost Problem

The computational results for this model show that the best known solution is always found by more than one procedure. Actually, three times all four algorithms obtained the best result. The Genetic Algorithm detected the minimal value of the objective function for all problems. Only for one problem, the Tabu Search and the Descent Procedure did not give the best known result.

A characteristic of the Total Cost Problem is that the value of p is obtained by the algorithms and not given in advance. For the best found solution of each problem it is presented in the table. However, concerning the number of assigned servers, there are no surprising results. Only for three problems, this value differs from the number determined by Beasley. This confirms that the parameters chosen for this model are appropriate.

Again the crucial question is, which algorithm is the most suited one to solve the underlying problem. In order to answer this question we will have a look at the results of all procedures.

Results of the Four Heuristics

Also for the Total Cost Problem the deviations from the best known objective function value are presented (see the following two tables Table 5.7 and Table 5.8). Again, all these values are expressed as a percentage over the best known (BK) result from Table 5.6.

	Descent			Simulated Annealing		
	% over BK result			% over BK result		
n	*min*	*ave*	*max*	*min*	*ave*	*max*
100	0.00	0.16	0.22	0.00	5.84	10.74
100	0.00	0.00	0.00	0.00	3.52	6.58
200	0.00	0.00	0.00	0.58	3.53	8.06
200	0.00	0.00	0.00	0.00	2.17	6.66
300	0.00	0.32	0.44	0.99	8.42	15.67
300	0.00	0.07	0.29	2.85	8.49	12.99
400	0.00	0.00	0.04	5.49	11.02	18.02
400	0.00	0.02	0.03	3.76	9.02	13.49
500	0.01	0.03	0.03	5.69	14.44	27.71
500	0.00	0.00	0.05	2.86	11.21	22.51
Average	0.00	0.06	0.11	2.22	7.76	14.24

Table 5.7: Descent- and Simulated Annealing- Results for the Total Cost Problem

The obtained results are particularly interesting. Compared to the Multiple Server Problem, where the deviations are to some extent very high, the results for the Total Cost Problem are consistent.

In the majority of cases, the Genetic Algorithm and the Tabu Search detected only one result, the best known solution. Thus, their deviation from the best known solution is really low. On average it is 0% for the Genetic Algorithm and 0.01% in case of the Tabu Search.

Also the results given by the Descent Procedure show a very small deviation from the best known result (on average 0.06%). Again, the Simulated Annealing Algorithm performed most insufficiently. Its deviation is on average 7.76% over the best known solution.

	Genetic Algorithm			Tabu Search		
	% over BK result			% over BK result		
n	*min*	*ave*	*max*	*min*	*ave*	*max*
100	0.00	0.00	0.00	0.00	0.00	0.00
100	0.00	0.00	0.00	0.00	0.00	0.01
200	0.00	0.00	0.00	0.00	0.00	0.00
200	0.00	0.00	0.00	0.00	0.00	0.00
300	0.00	0.01	0.32	0.00	0.01	0.32
300	0.00	0.00	0.03	0.03	0.05	0.27
400	0.00	0.00	0.00	0.00	0.00	0.00
400	0.00	0.00	0.00	0.00	0.00	0.00
500	0.00	0.01	0.01	0.00	0.03	0.03
500	0.00	0.00	0.00	0.00	0.00	0.00
Average	0.00	0.00	0.04	0.00	0.01	0.06

Table 5.8: Genetic Algorithm- and Tabu Search- Results for the Total Cost Problem

Conclusions for the Total Cost Problem

Concerning the objective function value, the Descent, the Genetic Algorithm, and the Tabu Search obtained the best results for almost all problems that were investigated. Only the performance of the Simulated Annealing Algorithm was insufficient.

Taking not only the best known result, but all found solutions into consideration shows that the Genetic Algorithm gave the best results, because its average deviation from the best found solution is the smallest.

Conclusions

Summing up the computational results for the Multiple Server Problem and the Total Cost Problem, we conclude that the Genetic Algorithm performs very well for both problems. Therefore, it is recommended to use this algorithm for solving our models. The Tabu Search is second in quality and thus also a good choice.

However, the running times of the algorithms are not considered. One has to keep in mind that the Genetic Algorithm takes the longest time to deliver results. As already mentioned, the running times of the Descent Procedure are on average under one second. Therefore, this algorithm is also an appropriate alternative since it can be executed many times and the probability that the best solution is found increases.

5.2 Sensitivity Analysis

In the second part of our experiments, the effects of different values of parameters such as the arrival rate, service rate, fixed installation costs of facilities, and variable server costs on the results are tested.

Beasley's first problem (pmed1) with 100 nodes and 5 servers is chosen to perform the analysis on. In case of the Total Cost Problem, the given number of servers is again neglected.

For this sensitivity analysis, the Descent Procedure is used. Although its performance is not among the best, there are several reasons for choosing this algorithm. First of all, this procedure is used by all the other algorithms and thus provides, to some extent, a basis for them. Secondly, its running times are the smallest. Moreover, changes in the parameters will have the same effects on the results for all procedures, even if their performance is not the best. Therefore, the Descent Procedure is used for the underlying sensitivity analysis.

Since the Total Cost Problem considers more components in the objective function, this model is particularly interesting for the sensitivity analysis. Thus, most of our analyses concentrate only on the Total Cost Problem.

Nevertheless, when varying the demand and service rate of the system, there are interesting differences between the Multiple Server Problem and the Total Cost Problem. Concerning these parameters, the sensitivity analysis is conducted for both models.

5.2.1 Varying Demand Rate and Service Rate

Besides minimizing the total cost, service stations will try to attract as many customers as possible. However, clients do not consider variable server costs or fixed installation costs. From their point of view, the values of the demand rate and the service rate are the most interesting ones. Their decision is very much dependent on these two values.

Thus, the ratio between the total capacity of service and the total demand of the customers in the system ($\frac{p\mu}{n\lambda}$) is varied. This ratio is denoted by the parameter θ (see Section 5.1.1).

For the computational experiments, θ was assumed to be 1.1. In the following, this parameter is varied between 1.01 and 1.2. θ being 1.01 leads to very tight service conditions, since the service capacity of all server units is only 1% more than the demand of all customers. Contrary to this, $\theta = 1.2$ represents very slack service capacity, because the total service capacity exceeds the total demand rate by 20%. As already mentioned, $\theta = 1.1$ leads to normal service conditions.

The Multiple Server Problem

In case of the Multiple Server Problem, the waiting costs at the stations are expected to decrease, when increasing the value of θ. The travel times of the customers to the stations are independent of θ. Thus, the total cost of the system is presumed to be a decreasing function in θ.

These assumptions can be checked in Table 5.9, where the results for different values of θ are presented.[2]

θ	minimal costs	waiting time	q
1.010	7890.00	5.000	5
1.025	8003.77	1.562	3
1.050	7309.98	0.786	3
1.075	6996.25	0.561	4
1.100	6981.48	0.420	4
1.125	6652.99	0.406	4
1.150	6641.36	0.311	4
1.175	6166.47	0.676	5
1.200	6116.00	0.425	5

Table 5.9: Varying θ in the Multiple Server Problem

As we can see, our a priori statements are true except for some outliers. For the value of θ being between 1.01 and 1.15, the waiting time is a decreasing function in θ. For the last two parameter values it is increasing again, which is due to a higher number of facilities. Considering the servers, one unit is placed at each station for θ being 1.01. In the following, servers are pooled at 3 or rather 4 facilities. As we have shown in Section 3.1, this also decreases the waiting times since more units serve the customer demand. For the last two values of θ, there is again just one server at each station.

[2]Note that only the best known results are given in this table.

Summing up, the total cost is decreasing in θ. This means that with a higher relative service rate (compared to the total demand rate), the total cost of the system diminishes.

Figure 5.1 presents a summary of the analysis with respect to the ratio between the total service and the total demand in the system. For this illustration, the ratio in total cost, waiting time, and number of facilities for each value of parameter is measured compared to the "base case" parameter value (i.e., $\theta = 1.1$). In case of the waiting time, this ratio is given by: $waiting\ time\ ratio\ (parameter\ value) = \frac{waiting\ time\ (parameter\ value)}{waiting\ time\ (base\ case\ parameter\ value)}$.

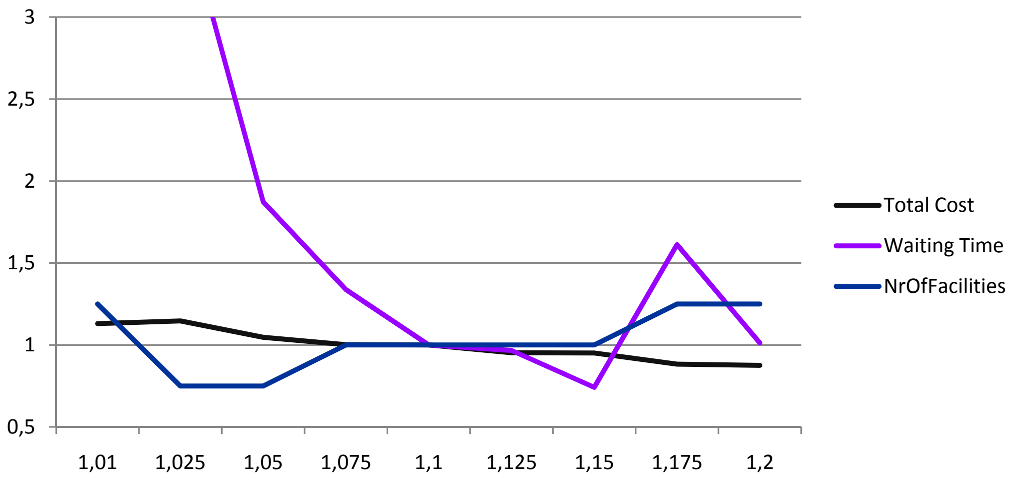

Figure 5.1: Sensitivity Analysis with respect to θ for the Multiple Server Problem

To sum up, in the Multiple Server Model, the costs are decreasing in the ratio between the total capacity of service and the total demand in the system $\theta = \frac{p\mu}{n\lambda}$.

The number of facilities in the system diminishes at first, but increases rapidly. Except for one outlier at the end (i.e., at θ being 1.175), the waiting times are also decreasing in θ. This outlier can be explained as follows: For θ being 1.175 the number of facility sites is equal to the number of servers in the system (i.e., 5). Thus only one service unit is placed at each facility node. As we have seen in Chapter 3 (see Section 3.1), the pooling of servers reduces the waiting times without additional costs. Consequently, the waiting times increase drastically when the number of servers is reduced to only one at each station.

The Total Cost Problem

For the Total Cost Problem, the waiting costs are also expected to be a decreasing function in θ. However, for all the other objective function components (the travel times, the fixed installation costs, and the variable server costs) it is not safe to make any assumptions.

Different values of θ might affect the total number of assigned servers, on which there is no upper limit in this model. With an increase in θ, a smaller number of server units is presumed. Thus, also the location of the facilities might change. Consequently, the variation of θ might affect customers' travel time, the fixed installation costs, and the variable server costs. Therefore, in case of the Total Cost Problem, the total cost of the system cannot be presumed to be a decreasing function in θ.

Table 5.10 represents the results for different values of θ in case of the Total Cost Problem.[3]

θ	minimal costs	waiting time	p	q
1.010	10257.83	0.130	6	2
1.025	10257.03	0.121	6	2
1.050	10255.93	0.109	6	2
1.075	10255.07	0.098	6	2
1.100	10254.36	0.091	6	2
1.125	10243.03	0.265	5	2
1.150	10222.35	0.223	5	2
1.175	10214.78	0.164	5	2
1.200	10210.91	0.133	5	2

Table 5.10: Varying θ in the Total Cost Problem

The waiting time is a decreasing function until θ being 1.1. Thus, the waiting costs of the customers diminish until this value. For higher parameter values, the number of assigned servers is reduced. Compared to 6 servers for values of θ being smaller than 1.125, only 5 units are assigned for θ being larger. Consequently, the waiting time increases for this particular parameter value and decreases afterwards again. Since the number of servers is diminished only by one at $\theta = 1.125$, the fixed server costs also become smaller and remain the same afterwards.

[3]Note that again only the best known results are given in this table.

For all values of θ, the number of facility sites is 2. Thus, there are no changes in the fixed installation costs.

Summing up these observations, the total cost in the Total Cost Problem are slowly decreasing in θ.

Again a summary of the analysis with respect to the ratio between the total service and the total demand in the system is presented (see Figure 5.2). The ratios in total cost, waiting time, number of facilities, and number of servers for all values of θ compared to the "base case" value (θ=1.1) are illustrated.

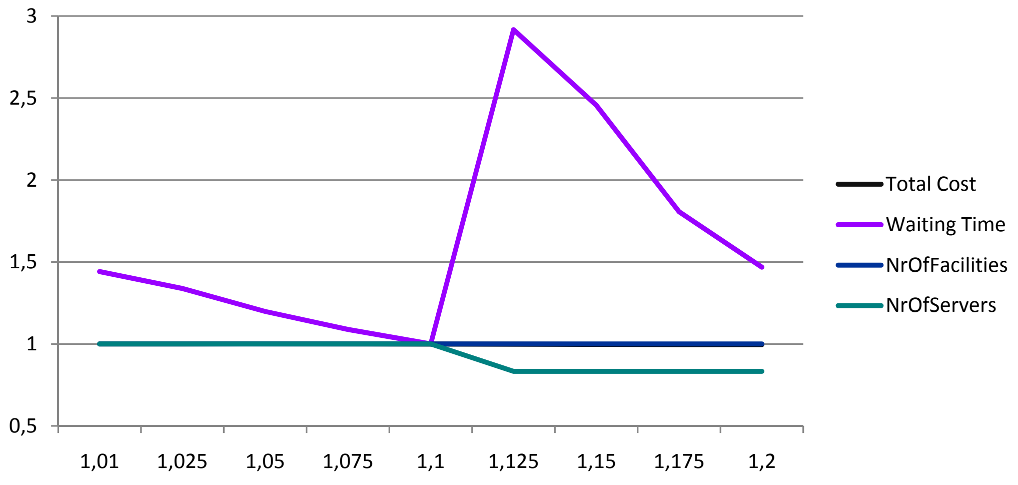

Figure 5.2: Sensitivity Analysis with respect to θ for the Total Cost Problem

Due to the fact that the total cost ratio is only very slowly decreasing in $\theta = \frac{p\mu}{n\lambda}$ and close to 1, it is hidden behind the ratios of the number of servers and of the number of facilities.

The figure clearly shows that the waiting time is increasing dramatically, when the number of servers is decreased.

The following two analyses (variation of the variable server costs h and the fixed installation costs f) are only conducted for the Total Cost Problem, since these parameters are not considered in the Multiple Server Problem.

5.2.2 Varying Variable Server Costs h

The focus in this section lies on the consequences of varying variable server costs h in the Total Cost Problem.

On the one hand, if the total number of assigned servers remains the same, the total cost will increase. On the other hand, due to higher server costs, fewer service units might be assigned to the stations. Because of a smaller number of servers, fewer facilities might be opened. In this case, the waiting times in the system and also the travel times increase. These facts also cause an increase in the total cost of the system.

It cannot be presumed, which factor influences the total cost of the system most. However, it is safe to say that the objective function value is an increasing function in h.

In the computational experiments, the variable server costs h were set to 50. Varying h between 20 and 100 leads to the following results represented in Table 5.11.[4]

h	minimal costs	waiting time	p	q
20	10074.36	0.0907	6	2
30	10134.36	0.0907	6	2
40	10194.36	0.0907	6	2
50	10254.36	0.0907	6	2
60	10312.04	0.3273	5	2
70	10362.04	0.3273	5	2
80	10412.04	0.3273	5	2
90	10462.04	0.3273	5	2
100	10512.04	0.3273	5	2

Table 5.11: Varying the Variable Server Costs h in the Total Cost Problem

The number of assigned servers remains the same for the first four values of h. Subsequently, it is minimized to five servers being in the system. Because of this, the waiting time is an increasing function in the variable server costs. The number of facilities remains the same for all values of h. Consequently, due to higher server costs and increasing waiting times, the total cost of the system also increases in the variable server costs.

[4]Like in the case of varying θ, only the best known results are presented.

Figure 5.3 presents a summary of the analysis with respect to the variable server costs. Again, the ratio in total cost, waiting time, number of facilities, and number of servers in the system for each value of parameter is measured compared to the "base case" parameter value (i.e., $h = 50$ in this case).

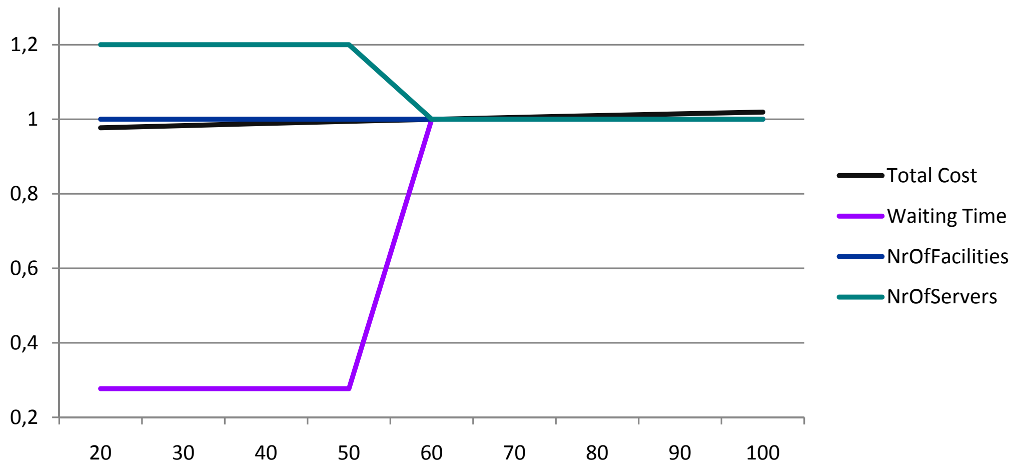

Figure 5.3: Sensitivity Analysis with respect to the Variable Server Costs h in the Total Cost Problem

This figure shows again that the waiting times increase drastically, when the number of servers is reduced. Since the total number of installed facilities in the system remains the same for all values of h, the ratio compared to the "base case" parameter equals one for all cases. The total cost is slowly increasing, when the variable server costs rise.

5.2.3 Varying Fixed Installation Costs f

If the number of facility stations does not change, an increase in the fixed installation costs f implicates rising total cost. However, it is quite safe to say that the number of service stations will not remain the same in case of a sufficient increase in f. But how does this number change? Are there also any effects on the number of servers and consequently on the waiting times in the system?

In order to answer these questions, the effects of increasing fixed installation costs on the total cost, the waiting times, the number of servers, and the number of facility stations are presented in Table 5.12.[5]

f	minimal costs	waiting time	p	q
200	6687.10	0.1015	9	9
400	8153.84	0.1677	7	7
600	9048.30	0.1426	6	4
800	9812.87	0.1462	6	3
1000	10254.36	0.0907	6	2
1200	10654.36	0.0907	6	2
1400	11054.36	0.0907	6	2
1600	11454.36	0.0907	6	2
1800	11854.36	0.0907	6	2
2000	12254.36	0.0907	6	2

Table 5.12: Varying the Fixed Installation Costs f in the Total Cost Problem

As assumed above, the number of service sites q decreases drastically with increasing fixed installation costs f. For f being 200, nine facilities are installed, in contrast to 2 service stations when f is 2000. Consequently, also the total number of servers in the system diminishes. Since the waiting time is very low, its influence on the total cost is not worth mentioning.

However, how can the decrease in the waiting time be explained if the number of servers p and the number of facilities q decrease? In Chapter 3 (see Section 3.1), we have seen that the pooling of servers decreases the waiting time of the customers. In case of varying fixed installation costs, this is also the case. Even if the number of facilities is decreasing, the waiting time is getting smaller because more servers are placed at the facility sites.

[5]Again only the best known results are presented in this table.

Although the two decision variables (the number of facilities and the number of assigned servers) are reduced, the total cost of the system is an increasing function in the fixed installation costs f.

Summing up, Figure 5.4 presents the analysis with respect to the fixed installation costs of a facility site.

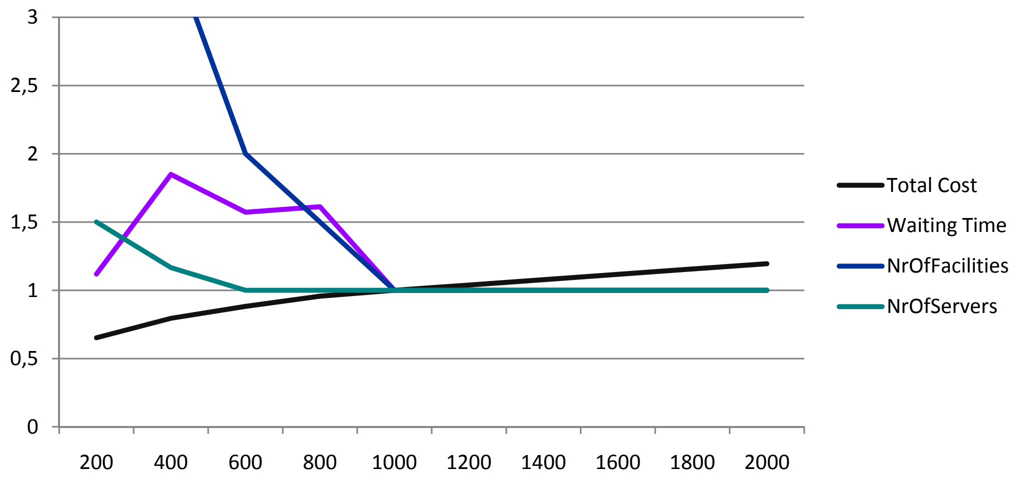

Figure 5.4: Sensitivity Analysis with respect to the Fixed Installation Costs f in the Total Cost Problem

The number of server units and the number of installed facilities is a decreasing function in the fixed installation costs f. However, the total cost of the system increase due to higher installation costs.

Chapter 6

Conclusions and Extensions for Further Research

6.1 Conclusions

This thesis was devoted to Facility Location Models with Stochastic Demand and Congestion. The primary focus was on the formulation of optimization models for the location-allocation problem that can be solved by heuristic search techniques.

After formulating a general model for Facility Location with stochastic customer demand and immobile servers, the Multiple Server Problem was introduced. For a given number of servers, the optimal location was to be found in order to minimize the travel and waiting times of all customers. This model is based on Berman and Drezner (2007), who analyzed the Multiple Server Location Problem. The difference to the model discussed in this thesis lies in the waiting times. While Berman and Drezner use the waiting time in the queue $W_q(\lambda, \mu, k)$, the waiting time in the system $W(\lambda, \mu, k)$ (i.e., waiting time in the queue plus service time) was considered here.

In addition, the Total Cost Problem was discussed. Aboolian et al. (2008) introduced this model, where the total cost of the system is sought to be minimized. Besides travel and waiting time, total cost also consists of fixed installation costs for facilities and of variable server costs. For this model, the number of servers in the system is not limited.

Four solution procedures (Descent Procedure, Simulated Annealing, Genetic Algorithm, and Tabu Search) were formulated, and examinations for the Multiple Server Problem and the Total Cost Problem were carried out.

In contrast to Aboolian et al. (2008), who only concentrated on the Descent Procedure and the Simulated Annealing Algorithm, we executed all four algorithms for the Total Cost Problem as well.

The Genetic Algorithm provided the highest quality results for both models. It detected the best solutions for most of the problems and the found results were always among the best of all heuristics. Therefore, the Genetic Algorithm is recommended for solving the underlying models. This coincides with the conclusions drawn in Berman and Drezner (2007). However, contrary to their recommendation not to use the Tabu Search, this algorithm performed well in our examination. Thus, the Tabu Search is also a good choice for solving the underlying problems from our point of view. While the Simulated Annealing Algorithm is recommended for normal service problems (i.e., $\theta = 1.1$) in Berman and Drezner (2007), its results in our experiments are not satisfying. The fact that only 10 out of the 40 Beasley problems were used in the underlying computational experiments might cause these different conclusions.

Finally, a sensitivity analysis was conducted in this thesis. The ratio between the total service capacity and the total demand in the system was varied for both models. For the Total Cost Problem, the consequences of a variation in the fixed installation costs and in the variable server costs were also analyzed.

6.2 Extensions for Further Research

By pointing to some possible extensions, this thesis is concluded. In the process of writing, various ideas and options came up that would allow going a step further than the analysis presented here. Therefore, it is recommended to tackle the following ideas and approaches, which could extend the underlying work concerning Stochastic Facility Location Models in several ways:

- In the models presented in this thesis it was assumed that customers visit the closest facility site. As already mentioned in Section 2.3.1, the assumption that clients have full information about the waiting time at the stations leads to complex formulations of the model. However, the model becomes more plausible. It is realistic that customers have information about the congestion at different stations. Therefore, not only the distance to the facility, but also the expected waiting time should be taken into consideration when choosing the facility.

Robert Aboolian (personal communication) is currently working on models, which consider customer traveling to the facility with the least access time (travel time and waiting time).

- Furthermore, there is a need for improving the solution procedures for the underlying models. The performance of the implemented heuristics for problems with a large number of nodes and of servers is insufficient. There should be ways to improve this.
- Another suggestion is to develop an exact solution algorithm. Aboolian et al. (2008) have already done this for the Total Cost Problem. Only small changes may be required when applying their results to the Multiple Server Problem.
- Since companies usually have a limited budget, not only the number of servers, but also the number of facilities might be bounded. Therefore, another topic for further investigation is to add this constraint in the model formulation.
- Additionally, for more realistic models, it is also reasonable to take the location of the competitors into consideration. One might lose customers, if the travel time to a facility of the competitor or the expected waiting time is assumed to be smaller.

Appendix A

Appendix

A.1 Efficient Calculations of W and W_q

In the case of an $M/M/k$ queueing system, the average waiting time in the queue $W_q(\lambda,\mu,k)$ and the average waiting time in the system $W(\lambda,\mu,k)$ are given by (see (3.2) and (3.3))

$$W_q(\lambda,\mu,k) = \frac{\left(\frac{\lambda}{\mu}\right)^k \rho}{k!\,(1-\rho)^2\,\lambda} P_0, \tag{A.1}$$

$$W(\lambda,\mu,k) = W_q(\lambda,\mu,k) + \frac{1}{\mu}, \tag{A.2}$$

where $\rho = \frac{\lambda}{k\mu}$.

These formulas are based on the probability that zero customers are in the system (see formula (3.1))

$$P_0(k) = \left[\sum_{n=0}^{k-1} \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} + \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!}\frac{1}{1-\rho}\right]^{-1}.$$

For large values of $\frac{\lambda}{\mu}$ (the ratio of the arrival rate to the service rate) Pasternack and Drezner (1998) point out that difficulties might arise when calculating $P_0(k)$. As $\frac{\lambda}{\mu}$ gets bigger, $P_0(k)$ converges towards zero. Due to computer underflow, the value of $P_0(k)$ might be calculated as zero. To avoid this problem, the authors formulate an efficient way of calculating the average number of customers in the queue $L_q(\lambda,\mu,k)$. Since we are interested in the waiting times, their suggestions are modified.

For a given λ and μ, a_k is defined as follows:

$$a_k = \rho \sum_{i=0}^{k-1} \frac{k!}{i!} \left(\frac{\lambda}{\mu}\right)^{i-k}.$$

Since $a_1 = 1$ and

$$\begin{aligned}
a_{k+1} &= \frac{\lambda}{\mu(k+1)} \sum_{i=0}^{k} \frac{(k+1)!}{i!} \left(\frac{\lambda}{\mu}\right)^{i-(k+1)} \\
&= \frac{k}{k+1} \frac{(k+1)\mu}{\lambda} \left[\frac{\lambda}{k\mu} \sum_{i=0}^{k} \frac{k!}{i!} \left(\frac{\lambda}{\mu}\right)^{i-k} \right] \\
&= \frac{k\mu}{\lambda} \left[a_k + \frac{\lambda}{k\mu} \right] \\
&= \frac{1}{\rho} a_k + 1,
\end{aligned}$$

the sequence a_k can be calculated recursively by

$$a_1 = 1; \qquad a_k = 1 + \frac{\mu}{\lambda}(k-1)a_{k-1}.$$

The definitions of $P_0(k)$ and a_k lead to the following:

$$\begin{aligned}
P_0(k) &= \left[\sum_{i=0}^{k-1} \frac{\left(\frac{\lambda}{\mu}\right)^i}{i!} + \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} \frac{1}{1-\rho} \right]^{-1} \\
&= \left[\frac{1}{\rho} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} a_k + \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} \frac{1}{1-\rho} \right]^{-1} \\
&= \frac{\frac{k!}{(\lambda/\mu)^k}}{\frac{1}{\rho} a_k + \frac{1}{1-\rho}}.
\end{aligned}$$

We would like to get a simple formula for calculating the total waiting time in the queue $W_q(\lambda, \mu, k)$ and in the system $W(\lambda, \mu, k)$. Therefore, the obtained result for the probability that zero customers are in the system, $P_0(k)$, is inserted into (A.1) and (A.2).

Consequently, $W_q(\lambda, \mu, k)$ can be calculated as follows

$$\begin{aligned}
W_q(\lambda, \mu, k) &= \frac{\left(\frac{\lambda}{\mu}\right)^k \rho}{k!(1-\rho)^2\lambda} \frac{\frac{k!}{(\lambda/\mu)^k}}{\frac{1}{\rho}a_k + \frac{1}{1-\rho}} \\
&= \frac{\rho}{(1-\rho)^2\lambda\left(\frac{1}{\rho}a_k + \frac{1}{1-\rho}\right)} \\
&= \frac{\rho}{(1-\rho)^2\lambda\frac{1}{\rho}\left(a_k + \frac{\rho}{1-\rho}\right)}, \quad \text{where } \rho = \frac{\lambda}{k\mu} \\
&= \frac{\frac{\lambda}{k\mu}}{\frac{(k\mu-\lambda)^2}{(k\mu)^2}\lambda\frac{k\mu}{\lambda}\left(a_k + \frac{\lambda}{k\mu-\lambda}\right)} \\
&= \frac{\lambda}{(k\mu-\lambda)^2\left(a_k + \frac{\lambda}{k\mu-\lambda}\right)}.
\end{aligned}$$

By adding $\frac{1}{\mu}$, a simple and efficient formula for $W(\lambda, \mu, k)$ is obtained:

$$W(\lambda, \mu, k) = \frac{\lambda}{(k\mu-\lambda)^2\left(a_k + \frac{\lambda}{k\mu-\lambda}\right)} + \frac{1}{\mu}.$$

A.2 Properties of W_q in an $M/M/k$ Queueing System

The average waiting time in the queue $W_q(\lambda, \mu, k)$ of an $M/M/k$ queueing system can be calculated by formula (A.1). In the following, it is assumed that the arrival rate λ and the service rate μ are given and W_q is only a function of k. Therefore, we just write $W_q(k)$.

The waiting time in the queue $W_q(k)$ is characterized by the following two properties:

Property 1: $W_q(k)$ is a decreasing function in the number of servers k.

Property 2: $W_q(k)$ is a convex function in the number of servers k.

For the proof of these properties, two propositions are needed (see Dyer and Proll, 1977).

Proposition 1
The function $H(k) = \frac{F(k)}{G(k)}$ is strictly decreasing in k if $F(k)$ is positive and strictly decreasing in k and $G(k)$ is positive and strictly increasing in k.

Proof: For a function $f(k)$ the difference operator $\Delta f(k)$ denotes the difference $f(k+1) - f(k)$. In case of an increasing function, the difference operator is always positive. For a decreasing function it is always negative.

Thus, the conditions of the Proposition are equivalent to:

$$F(k), G(k), \Delta G(k) > 0 \text{ and } \Delta F(k) < 0. \tag{A.3}$$

We have to show that $\Delta H(k) < 0$.

$$\begin{aligned}\Delta H(k) &= \frac{F(k+1)}{G(k+1)} - \frac{F(k)}{G(k)} \\ &= \frac{F(k+1)G(k) - F(k)G(k+1)}{G(k+1)G(k)} \\ &= \frac{G(k)\Delta F(k) - F(k)\Delta G(k)}{G(k+1)G(k)}\end{aligned}$$

According to our assumptions, this fraction is negative. □

Now this result is applied to the average waiting time in the queue $W_q(k)$ (see (A.1)) in an $M/M/k$ queueing system. Making some transformations leads to the following term:

$$W_q(k) = \frac{\varsigma^k}{(k-1)!\,(k-\varsigma)\,\mu}\left[(k-\varsigma)\sum_{n=0}^{k-1}\frac{\varsigma^n}{n!} + \frac{\varsigma^k}{(k-1)!}\right]^{-1},$$

where $\varsigma = \frac{\lambda}{\mu}$.

Thus, $F(k)$ and $G(k)$ in $W_q(k) = \frac{F(k)}{G(k)}$ have the following form:[1]

$$F(k) = \frac{\varsigma^k}{(k-1)!\,(k-\varsigma)} \text{ and } G(k) = (k-\varsigma)\sum_{n=0}^{k-1}\frac{\varsigma^n}{n!} + \frac{\varsigma^k}{(k-1)!}. \tag{A.4}$$

It is clear that the first two conditions, $F(k) > 0$ and $G(k) > 0$, are fulfilled, since $k > \varsigma > 0$. What about $\Delta G(k)$ and $\Delta F(k)$? Do they also meet the conditions?

$$\begin{aligned}\Delta G(k) &= (k+1-\varsigma)\sum_{n=0}^{k}\frac{\varsigma^n}{n!} + \frac{\varsigma^{k+1}}{k!} - (k-\varsigma)\sum_{n=0}^{k-1}\frac{\varsigma^n}{n!} - \frac{\varsigma^k}{(k-1)!}\\ &= \sum_{n=0}^{k-1}\frac{[(k+1-\varsigma)-(k-\varsigma)]\,\varsigma^n}{n!} + \frac{(k+1-\varsigma)\,\varsigma^k}{k!} +\\ &\quad + \frac{(\varsigma-k)\,\varsigma^k}{k!}\\ &= \sum_{n=0}^{k-1}\frac{\varsigma^n}{n!} + \frac{\varsigma^k}{k!} = \sum_{n=0}^{k}\frac{\varsigma^n}{n!} > 0,\end{aligned} \tag{A.5}$$

$$\begin{aligned}\Delta F(k) &= \frac{\varsigma^{k+1}}{k!\,(k+1-\varsigma)} - \frac{\varsigma^k}{(k-1)!\,(k-\varsigma)}\\ &= \frac{\varsigma^k}{k!}\left[\frac{\varsigma}{k+1-\varsigma} - \frac{k\,(-\varsigma+\varsigma)}{k-\varsigma}\right]\\ &= \frac{\varsigma^k}{k!}\left[-\frac{\varsigma}{(k+1-\varsigma)\,(k-\varsigma)} - 1\right] < 0,\end{aligned} \tag{A.6}$$

since $k > \varsigma > 0$.

Thus, all constraints in (A.3) are met and $H(k) = W_q(k)$ is a decreasing function in k. Property 1 is therefore proven.

[1] $\frac{F(k)\frac{1}{\mu}}{G(k)}$ is a decreasing function in k if $\frac{F(k)}{G(k)}$ is decreasing. Thus, $\frac{1}{\mu}$ can be skipped.

In order to prove Property 2, the following Proposition is needed:

Proposition 2
Let $F(k)$ and $G(k)$ be characterized by the conditions in (A.3).
$H(k) = \frac{F(k)}{G(k)}$ is a convex function in k if

$$F(k+1)\Delta^2 G(k) - G(k+1)\Delta^2 F(k) < 0. \tag{A.7}$$

PROOF: We prove the Proposition using the following equation:
$H(k)$ is convex if and only if $\Delta^2 H(k) > 0$.

$$-\Delta H(k) = \frac{F(k)\Delta G(k) - G(k)\Delta F(k)}{G(k+1)G(k)}$$

Both the enumerator and the denominator of this fraction $-\Delta H(k)$ are positive due to (A.3).

Let $E(k) = F(k)\Delta G(k) - G(k)\Delta F(k)$ and $D(k) = G(k+1)G(k)$.

$$\begin{aligned}
\Delta E(k) &= F(k+1)\Delta G(k+1) - F(k)\Delta G(k) \\
&\quad - G(k+1)\Delta F(k+1) + G(k)\Delta F(k) \\
&= F(k+1)\left[\Delta G(k+1) - \Delta G(k)\right] + \Delta G(k)\left[F(k+1) - F(k)\right] \\
&\quad - G(k+1)\left[\Delta F(k+1) - \Delta F(k)\right] - \Delta F(k)\left[G(k+1) - G(k)\right] \\
&= F(k+1)\Delta^2 G(k) - G(k+1)\Delta^2 F(k).
\end{aligned}$$

Due to property (A.7), $\Delta E(k) < 0$.

$$\begin{aligned}
\Delta D(k) &= G(k+2)G(k+1) - G(k+1)G(k) \\
&= G(k+1)\left[G(k+2) - G(k)\right].
\end{aligned}$$

Since $G(k)$ is strictly increasing, $\Delta D(k)>0$.

$E(k)$ and $D(k)$ satisfy the conditions in (A.3) and Proposition 1 can be applied: $-\Delta H(k) = \frac{\Delta E(k)}{\Delta D(k)} < 0$. Thus, $\Delta H(k) > 0$, $\Delta^2 H(k) > 0$, and $H(k)$ is convex. □

If the conditions in Proposition 2 are satisfied for $W_q(k)$, Property 2 is also proven. It only has to be shown that inequality (A.7) holds for $F(k)$ and $G(k)$.

Using (A.5), we get $\Delta^2 G(k) = \frac{\varsigma^{k+1}}{(k+1)!} > 0$.
For $F(k)$, (A.6) is used and the following is obtained:

$$\begin{aligned}
\Delta^2 F(k) &= -\left[1 + \frac{\varsigma}{(k+1-\varsigma)\,(k+2-\varsigma)}\right] \frac{\varsigma^{k+1}}{(k+1)!} \\
&\quad + \left[1 + \frac{\varsigma}{(k-\varsigma)\,(k+1-\varsigma)}\right] \frac{\varsigma^{k}}{k!} \\
&= \ldots \\
&= \frac{k+1-\varsigma}{(k+1)!}\varsigma^k + \left(1 + \frac{\varsigma+1}{k-\varsigma} - \frac{\varsigma}{k+2-\varsigma}\right) \frac{\varsigma^{k+1}}{(k+1)!\,(k+1-\varsigma)} \\
&\geq \left[1 + \varsigma\left(\frac{1}{k-\varsigma} - \frac{1}{k+2-\varsigma}\right)\right] \frac{\varsigma^{k+1}}{(k+1)!\,(k+1-\varsigma)} \\
&\geq \frac{\varsigma^{k+1}}{(k+1)!\,(k+1-\varsigma)} \geq 0.
\end{aligned}$$

Therefore,

$$\Delta^2 G(k) - (k+1-\varsigma)\Delta^2 F(k) \leq \frac{\varsigma^{k+1}}{(k+1)!} - \frac{\varsigma^{k+1}}{(k+1)!} = 0.$$

Using the definitions of $G(k)$ and $F(k)$ (see (A.4)) leads to the following:

$$\begin{aligned}
G(k+1) &= (k+1-\varsigma)\sum_{n=0}^{k} \frac{\varsigma^n}{n!} + \frac{\varsigma^{k+1}}{k!} \\
&\geq \frac{\varsigma^{k+1}}{k!} \\
&= (k+1-\varsigma)F(k+1).
\end{aligned}$$

With the help of these results, we obtain

$$\begin{aligned}
&F(k+1)\Delta^2 G(k) - G(k+1)\Delta^2 F(k) \\
&\qquad\qquad \leq F(k+1)\left[\Delta^2 G(k) - (k+1-\varsigma)\Delta^2 F(k)\right] \leq 0,
\end{aligned}$$

which proves that $W_q(k)$ is a convex function in k.

A.3 Implemented Algorithms

In Chapter 3, four algorithms (Descent Procedure, Simulated Annealing, Genetic Algorithm, and Tabu Search) are introduced and formulated in order to solve the two underlying models. These search procedures are coded in C#. The results from the computational experiences and the sensitivity analysis are presented in Chapter 4.

Including the source codes of all four algorithms for both models would be beyond the scope of this thesis. Therefore, only the implemented Genetic Algorithm for the Multiple Server Problem is presented as an example for all programs. This procedure is chosen, because it detected the best solutions for most of the problems, and its results are always among the best.

Implemented Genetic Algorithm for the Multiple Server Problem

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MSLP;
using System.IO;

namespace MSLPNew {
  // solve Multiple Server Problem or Total Cost Problem using a Genetic Algorithm
  class GeneticSolver : AbstractSolver {
    private int nrOfIterations;
    DescentSolver descentSolver;

    private bool totalCosts;
    private int serverCosts = 0;
    private int installationCosts = 0;
    private int waitingCosts = 0;
    private int travelCosts = 0;

    Solution[] mergeNeighbours;

    // constructor for the Multiple Server Problem
    public GeneticSolver(Distances distances, int nrOfEdges, int maxNrOfServers, double
     demandRate, double serviceRate, int nrOfIterations)
      : base(distances, nrOfEdges, maxNrOfServers, demandRate, serviceRate) {
      this.nrOfIterations = nrOfIterations;
      this.mergeNeighbours = new Solution[maxNrOfServers * 2];

    List<int> nodeRestrictions = new List<int>();
    for (int m = 0; m < nrOfNodes; m++) {
      nodeRestrictions.Add(m);
      if (m < maxNrOfServers * 2)
        mergeNeighbours[m] = new Solution(nrOfNodes, nodeShortestDistances, demandRate,
  serviceRate, maxNrOfServers, 0, 0, 0, 0);
    }

    this.totalCosts = false;

    descentSolver = new DescentSolver(distances, nrOfEdges, maxNrOfServers, demandRate,
  serviceRate, nodeRestrictions);
  }

  // constructor fot the Total Cost Problem
  public GeneticSolver(Distances distances, int nrOfEdges, int maxNrOfServers, double
  demandRate, double serviceRate, int nrOfIterations,
    int serverCosts, int installationCosts , int waitingCosts, int travelCosts)
    : base(distances, nrOfEdges, maxNrOfServers, demandRate, serviceRate) {
    this.nrOfIterations = nrOfIterations;
    this.mergeNeighbours = new Solution[maxNrOfServers * 2];

    List<int> nodeRestrictions = new List<int>();
    for (int m = 0; m < nrOfNodes; m++) {
      nodeRestrictions.Add(m);
      if (m < maxNrOfServers * 2)
        mergeNeighbours[m] = new Solution(nrOfNodes, nodeShortestDistances, demandRate,
  serviceRate, maxNrOfServers, serverCosts, installationCosts, waitingCosts, travelCosts);
    }

    this.totalCosts = true;
    this.serverCosts = serverCosts;
    this.installationCosts = installationCosts;
    this.waitingCosts = waitingCosts;
    this.travelCosts = travelCosts;

    descentSolver = new DescentSolver(distances, nrOfEdges, maxNrOfServers, demandRate,
  serviceRate, nodeRestrictions, serverCosts, installationCosts, waitingCosts, travelCosts);
  }

  public override Solution solveMSLP() {
    Solution[] population = new Solution[nrOfNodes];
    double[] costsOfPopulation = new double[nrOfNodes];
    int posOfWorstMember = −1;
    double highestCosts = double.MinValue;

    // create the initial population
    // ----------------------------------------
    for (int i = 0; i < nrOfNodes; i++) {
      population[i] = descentSolver.solveMSLP();
      double currCosts;
      if (totalCosts)
        currCosts = population[i].getTotalCosts();
      else
        currCosts = population[i].getCosts();
      costsOfPopulation[i] = currCosts;
      if (currCosts > highestCosts) {
        highestCosts = currCosts;
        posOfWorstMember = i;
      }
    }
    // ----------------------------------------

    Random rand = new Random();
    Solution mergeResult = null;
    for (int i = 0; i < nrOfIterations; i++) { // generate <nrOfIterations> generations
      // randomly select two individuals from the population
      int firstIndividual = rand.Next(nrOfNodes);
      int secondIndividual = rand.Next(nrOfNodes);
      while (secondIndividual == firstIndividual) {
        secondIndividual = rand.Next(nrOfNodes);
      }

      // merge these selected individuals and create a new individual
      mergeResult = merge(population[firstIndividual], population[secondIndividual]);
      if (mergeResult != null) {
        // assign the number of servers assigned to the chosen facilities
        if (totalCosts)
          mergeResult.assignNumberOfServersToPositionsTotalCosts();
        else
          mergeResult.assignNumberOfServersToPositions();
        double newCosts;
        // calculate the costs of the new individual
        if (totalCosts)
          newCosts = mergeResult.getTotalCosts();
        else
          newCosts = mergeResult.getCosts();
        // check if the fitness of the new individual is better than the fitness of the worst
    individual in the population
        if (newCosts < highestCosts) {
          // check if this solution already exists in the population
          if (!isSolutionInPopulation(mergeResult, population)) {
                // replace the worst member
                population[posOfWorstMember] = mergeResult;
                costsOfPopulation[posOfWorstMember] = newCosts;
                highestCosts = double.MinValue;
                // find and store the individual with the worst fitness
                for (int j = 0; j < nrOfNodes; j++) {
                  if (costsOfPopulation[j] > highestCosts) {
                    highestCosts = costsOfPopulation[j];
                    posOfWorstMember = j;
                  }
                }
              }
            }
          }
        }

        double bestCosts = highestCosts;
        int posOfBest = 0;
        // find the best member in the population
        for (int i = 0; i < nrOfNodes; i++) {
          if (costsOfPopulation[i] < bestCosts) {
            bestCosts = costsOfPopulation[i];
            posOfBest = i;
          }
        }
        // return the best member
        return population[posOfBest];
      }

      // check if a certain solution already exists in a particular population
      private bool isSolutionInPopulation(Solution mergeResult, Solution[] population) {
        int [] serversInSolution = mergeResult.getServers();
        for (int i = 0; i < nrOfNodes; i++) {
          bool isCurrEqual = true;
          for (int j = 0; j < mergeResult.nrOfServers; j++) {
            if (!population[i].containsServer(serversInSolution[j])) {
              isCurrEqual = false;
              break;
            }
          }
          if (isCurrEqual)
            return true;
        }
        return false;
      }

```

```
    // merge two individuals and create a new individual
    private Solution merge(Solution firstIndividual, Solution secondIndividual) {
      Solution intersection = new Solution(nrOfNodes, nodeShortestDistances, demandRate,
     serviceRate, maxNrOfServers, serverCosts, installationCosts, waitingCosts, travelCosts);
      List<int> difference = new List<int>();

      // calculate the intersection and the difference of the two given individuals
      getDifferenceAndIntersection(ref intersection, ref difference, firstIndividual,
     secondIndividual);

      if (difference.Count > 0) { // if the two individuals are equal, this parameter equals 0
        // run a restricted Descent Procedure
        descentSolver.setNodeRestrictions(difference);
        return descentSolver.solveMSLP(intersection);
      }
      return intersection;
    }

    // calculate the difference and the intersection of two given individuals
    private void getDifferenceAndIntersection(ref Solution intersection, ref List<int> difference,
      Solution firstIndividual, Solution secondIndividual) {
      int[] serversOfFirst = firstIndividual.getServers();
      int[] serversOfSecond = secondIndividual.getServers();
      for (int i = 0; i < firstIndividual.nrOfServers; i++) {
        int currServerOfFirst = serversOfFirst[i];
        if (secondIndividual.containsServer(currServerOfFirst))
          intersection.addServer(currServerOfFirst, false);

        if (!secondIndividual.containsServer(currServerOfFirst))
          difference.Add(currServerOfFirst);
      }
      for (int i = 0; i < secondIndividual.nrOfServers; i++) {
        int currServerOfSecond = serversOfSecond[i];
        if (!firstIndividual.containsServer(currServerOfSecond))
          difference.Add(currServerOfSecond);
      }
      int nrOfRandAdds = 0;
      Random rand = new Random();
      // randomly add 3 nodes to the difference (for the restricted Descent Procedure)
      // this represents the mutation of the Genetic Algorithm
      while (nrOfRandAdds < 3) {
        int addNode = rand.Next(nrOfNodes);
        if (!firstIndividual.containsServer(addNode) && !secondIndividual.containsServer(
     addNode)) {
          difference.Add(addNode);
          nrOfRandAdds++;
```

```
            }
         }
         intersection .assignClientsToServer();
      }
   }
}
```

Bibliography

Aboolian, R., Berman, O., and Drezner, Z. (2008). Location and Allocation of Service Units on a Congested Network. *IIE Transactions*, 40(4):422–433.

Aboolian, R., Berman, O., and Drezner, Z. (2009). The Multiple Server Center Location Problem. *Annals of Operations Research*, 167(1):337–352.

Alp, O., Erkut, E., and Drezner, Z. (2003). An efficient Genetic Algorithm for the p-Median Problem. *Annals of Operations Research*, 122(1):21–42.

Batta, R. and Berman, O. (1989). A Location Model for a Facility Operating as an M/G/k Queue. *Networks*, 19(6):717–728.

Batta, R., Dolan, J., and Krishnamurthy, N. (1989). The Maximal Expected Covering Location Problem: Revisited. *Transportation Science*, 23(4):277–287.

Batta, R., Larson, R., and Odoni, A. (1988). A Single Server Priority Queueing Location Model. *Networks*, 8:87–103.

Beasley, J. (1985). A Note on Solving Large p-Median Problems. *European Journal of Operational Research*, 21(2):270–273.

Beasley, J. (1990). OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Berman, O. and Drezner, Z. (2007). The Multiple Server Location Problem. *Journal of the Operational Research Society*, 58(1):91–99.

Berman, O. and Krass, D. (2002). Facility Location Problems with Stochastic Demands and Congestion. In Drezner, Z. and Hamacher, H., editors, *Facility Location: Applications and Theory*, chapter 11, pages 329–371. Springer Verlag, Berlin.

Berman, O., Krass, D., and Wang, J. (2006). Locating Service Facilities to Reduce Lost Demand. *IIE Transactions*, 38(11):933–946.

Berman, O., Larson, R., and Chiu, S. (1985). Optimal Server Location on a Network Operating as an M/G/1 Queue. *Operations Research*, 33(4):746–771.

Berman, O., Larson, R., and Parkan, C. (1987). The Stochastic Queue p-Median Problem. *Transportation Science*, 21(3):207.

Berman, O. and Mandowsky, R. (1986). Location-Allocation on Congested Networks. *European Journal of Operational Research*, 26(2):238–250.

Černý, V. (1985). Thermodynamical Approach to the Traveling Salesman Problem: An efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* D. Appleton, New York.

Daskin, M. (1983). Maximum Expected Covering Location Model: Formulation, Properties and Heuristic Solution. *Transportation Science*, 17(1):48–70.

Dawid, H. (1996). *Adaptive Learning by Genetic Algorithms: Analytical Results and Applications to Economic Models.* Springer-Verlag, Berlin.

Drezner, T., Drezner, Z., and Mihaylo, S. (2010). The Gravity Multiple Server Location Problem. *Working Paper, College of Business and Economics, California State University-Fullerton.*

Drezner, Z. (1988). On the Pooling of Services. *Interfaces*, 18(2):108–110.

Dyer, M. and Proll, L. (1977). On the Validity of Marginal Analysis for Allocating Servers in M/M/c Queues. *Management Science*, 23(9):1019–1022.

Felsenstein, E. and Stiermaier, S. (2009). A Literature Survey on Facility Location, the Fermat-Weber-Problem and Queueing-Location Problems. *Practical Thesis, Vienna University of Technology.*

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13(5):533–549.

Glover, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. and Kochenberger, G. (2003). *Handbook of Metaheuristics, International Series in Operations Research & Management Science.* Kluwer Academic Publishers, Norwell, MA.

Glover, F. and Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers, Norwell, MA.

Goldberg, D. (1989). *Genetic Algorithms in Search and Optimization.* Addison-Wesley Publishing Company, Massachusetts, USA.

Heaton, J. (2008). *Introduction to Neural Networks for C#.* Heaton Research, Chesterfield, MO, 2nd edition.

Heuser, H. (2004). *Lehrbuch der Analysis, Teil 2.* Teubner Verlag, Stuttgart, 13th edition.

Hillier, F. and Lieberman, G. (2005). *Introduction to Operations Research.* McGraw-Hill Science, Engineering & Mathematics, New York, 8th edition.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Larson, R. (1974). A Hypercube Queueing Model for Facility Location and Redistricting in Urban Emergency Services. *Computers and Operations Research*, 1:67–95.

Marianov, V. and ReVelle, C. (1994). The Queuing Probabilistic Location Set Covering Problem and Some Extensions. *Socioeconomic Planning Sciences*, 28:167–178.

Marianov, V. and ReVelle, C. (1996). The Queueing Maximal Availability Location Problem: a Model for the Siting of Emergency Vehicles. *European Journal of Operational Research*, 93(1):110–120.

Marianov, V. and Ríos, M. (2000). A Probabilistic Quality of Service Constraint for a Location Model of Switches in ATM Communications Networks. *Annals of Operations Research*, 96(1):237–243.

Marianov, V. and Serra, D. (1998). Probabilistic, Maximal Covering Location-Allocation Models for Congested Systems. *Journal of Regional Science-Philadelphia*, 38:401–424.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of State Calculations by fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

Pasternack, B. and Drezner, Z. (1998). A Note on Calculating Steady State Results for an M/M/k Queueing System when the Ratio of the Arrival Rate to the Service Rate is Large. *Journal of Applied Mathematics and Decision Sciences*, 2(2):201–203.

ReVelle, C. and Hogan, K. (1989). The Maximum Availability Location Problem. *Transportation Science*, 23(3):192–200.

Ribeiro Filho, J., Treleaven, P., and Alippi, C. (1994). Genetic-Algorithm Programming Environments. *IEEE Computer*, 27(6):28–43.

Wang, Q., Batta, R., and Rump, C. (2002). Algorithms for a Facility Location Problem with Stochastic Customer Demand and Immobile Servers. *Annals of Operations Research*, 111(1):17–34.

Wang, Q., Batta, R., and Rump, C. (2004). Facility Location Models for Immobile Servers with Stochastic Demand. *Naval Research Logistics*, 51(1):137–152.

List of Algorithms

List of Figures

List of Tables