Bachelor Thesis Econometrie & Besliskunde 2009

Clustering in SaaS networks

Jelmer van der Gaast 303563

Niels Rietveld 302886

June 20, 2009

Supervisor: Dr. A.F. Gabor

Abstract

In this thesis the effects of clustering in SaaS networks were investigated. With use of clustering the costs of a SaaS company can be reduced. The assumption is made, that all the requested applications of an organization has to be placed on only one cluster but several organizations can be dedicated to a cluster. The first part of this research was to formulate an IP model for minimizing the total costs of clustering. However, due to computational limits this formulation can not directly be solved optimally by CPLEX and therefore several heuristics were developed. Of these heuristics, Tabu Search performed better than both the Lagrange heuristic and the Genetic Algorithm.

In the second part of this research the effects of stochastic demand were determined. In this case, stochastic demand is incorporated in the IP formulation and the total (expected) costs are minimized for two-stages. The scenarios in the second-stage were constructed by a procedure inspired by SAA. Tabu Search and Genetic Algorithm were adapted to this extension. Again, Tabu Search gave the best results, however the in large data sets GA ran much faster.

Declaration

The work in this thesis is based on research carried out at the Erasmus School of Economics, The Netherlands. No part of this bachelor thesis has been submitted elsewhere for any other degree or qualification and it all our own work unless referenced to the contrary in the text.

Acknowledgments

Special thanks to our supervisor Dr. A. F. Gabor who proposed the idea of this thesis and supported us during the writing process.

Contents

List of Tables

List of Figures

1 Introduction

Clustering in *Software as a Service* (SaaS) networks is investigated in this thesis. SaaS software are computer applications which are not hosted in an enterprise or at a desktop, but are hosted out of the enterprise. SaaS companies have to install these applications on their servers, which are called clusters. A SaaS client can then access these applications via the web.

By clustering is meant to merge different organizations to one and the same cluster in a SaaS network. This can be beneficial for the SaaS company, because it can reduce the costs of hardware and installing software. Clustering might reduce the total necessary clusters and if SaaS clients request the same applications, the SaaS company has to install this application only once. SaaS networks consist of several clusters and all clients of the SaaS company have to be assigned to a cluster. The clusters differ in installed applications and size. The size depends on the number of nodes in a cluster. In this thesis it is investigated how clustering in SaaS networks can be carried out in the most efficient way. It is investigated how the total costs for the SaaS company can be minimized.

The relevance of this subject is beyond all doubt. The market for SaaS software is rapidly growing. The revenue of SaaS in 2009 is expected to be $7.1 billion and that till 2013 this market will grow every year with at least 20%. This indicates the importance of this research. Not only for determining the optimal clustering for now, but also for determining the optimal clustering with respect to the future.

The outline of this thesis is therefore as follows. First a literature study is carried out. In this part the idea behind SaaS is explained. After that, some articles about the *Facility Location Problem* (FLP) are discussed, since our *SaaS Clustering Problem* (SCP) is formulated in a similar way. In several articles, Tabu Search and Genetic Algorithm were applied to find a solution of the FLP; these articles are shortly described in separate sections. The last part of the literature study consists of a short introduction to Stochastic Integer Programming. This is important for determining the optimal cluster configuration when the demand is not known for the future.

In the second part of this research, the IP formulation of the SCP is given and the

Lagrangian relaxation of this formulation is derived. After that, both heuristics — Tabu Search and Genetic Algorithm — are introduced and applied to the SCP. At the end, a comparison between these two heuristics is given.

The stochastic extension of SCP is described in the third part. In the *Stochastic SCP* (SSCP) demand can differ in the second stage and therefore several scenarios has to be considered. The total costs are then minimized for two stages. In addition, Tabu Search and Genetic Algorithm are adapted to this new formulation.

Part four shows the computational results which are obtained for the test cases. The creation of the test cases are described in detail. The results of the SCP and the SSCP are investigated and some guidelines for clustering are derived from the obtained results. These guidelines give rules of thumb for clustering in SaaS networks.

The conclusion of the research is given in the last and fifth section. The conclusion gives a drawback to the used methods for clustering in SaaS networks. Some remarks on the research will be given along with some further research topics.

2 Literature research

2.1 SaaS general

Since early 90's there has been a rapid advance in the Internet in terms of speed, connectivity and reliability (Nitu, 2009). With the Internet becoming everywhere available, coupled with the vendor's interest in capturing the market of small customers who could not afford the expensive enterprise software, the rise of *Software as a Service* (SaaS) has been fueled. A working definition of SaaS is given by de Jong (2009):

> "SaaS delivers software as a service to the customer through the Internet, the software is managed centrally, delivered and owned by one or more providers in such way that the customers has not the burden of maintaining the software they use."

SaaS is thus a software delivery paradigm where the software is hosted out of the enterprise and delivered via the web. The organization is connected via the Internet with the SaaS company. The end user in the company requests for applications via the Internet. These applications are supplied from the server in the SaaS company through the Internet to the end user. This is shown graphically in figure 2.1.

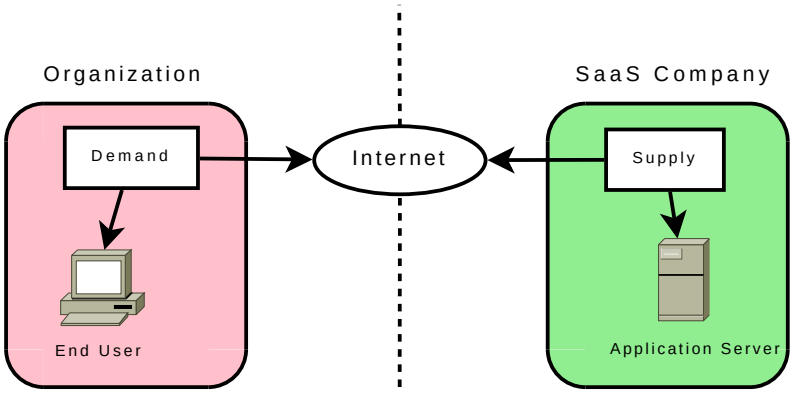


Figure 2.1: Software as a Service design

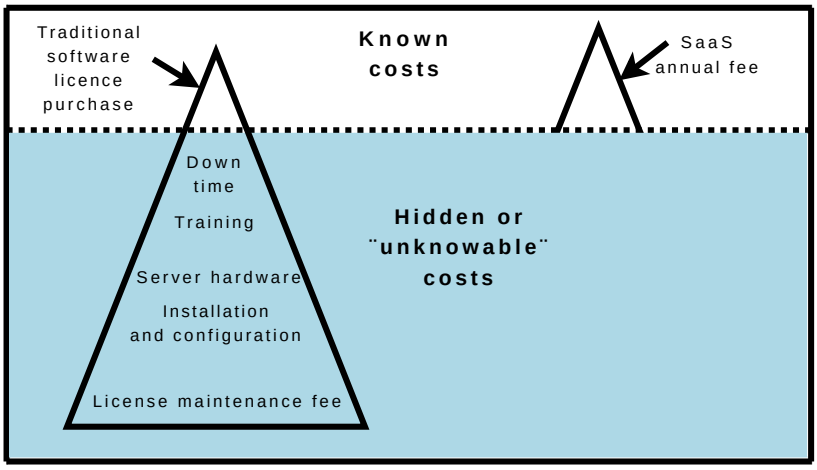


Figure 2.2: Client costs in SaaS (Waters, 2005)

It was only in 2005-2006 that the SaaS wave gained momentum. The reason for this was that not only the Internet had become high-speed and affordable, but also the customers had started becoming comfortable with doing business on the web. It was realized that good backup and fault tolerant practices at a good SaaS vendor may make data more secure and reliable with the SaaS company rather than with own enterprise. Since the number of SaaS vendors had increased, the customers had the advantage of flexibility in making their IT decisions as they had not invested in the infrastructure and could always shift to another vendor if needed. Due to the faster upgrade cycle of SaaS software and small deployment time, the customers need not spend time and money in upgrades and deployment of the software. These cost-advantages are shown in figure 2.2.

The mode of payment follows a subscription mode. The enterprise which uses SaaS software has to pay the (annual) fee to the SaaS company for using the software. These are roughly the only IT-costs the enterprise has. In a company which uses software in a traditional way, the actual IT-costs are more hidden. The costs of software licenses are known, but costs of down time, training of IT-personnel, server hardware, installation, upgrading and the license maintenance fee are difficult to predict. An enterprise which uses SaaS software does not have to make large investments in hardware and training of IT-personnel before they can use the software. They pay only a fee to the SaaS company for using software. This fee is higher than the costs of traditional software licenses, but now they face not longer (substantial) other hidden IT-costs. On the SaaS company's side, not only they could tap newer markets through SaaS, they also found consistent periodic revenues from the subscription model of SaaS very attractive (Nitu, 2009).

In the Netherlands enterprises like Cordys, SaaS Nederland and Microsoft deliver SaaS software. The market for SaaS software is rapidly growing. The revenue of SaaS in 2009 is expected to be $7.1 billion worldwide (Kivits, 2009). This is an increase of $2.5 billion compared with the revenue in 2008. It is expected that till 2013 this market will grow every year with at least 20%.

2.2 Facility Location Problem

The SaaS Clustering Problem developed in this thesis shares many similarities with the widely known *Capacitated Facility Location Problem* (CFLP). The CFLP is a well studied NP-hard combinatorial optimization problem (Ghiani et al., 2004), for which several solution methods have been developed the past fifty years. CFLP is a generalization of the *Simple Facility Location Problem* (SFLP). In the CFLP facilities (warehouses, facilities, plants, etc.) are located optimal in such a way that the costs associated with opening plants and serving the customers are minimized and the capacity in a facility is limited.

In the last ten years, some new extensions for the CFLP were developed. For example, general cost functions and the possibility to open several facilities on one location. The following articles addresses one or more of these extensions.

Hajiaghayi et al. (2002) created a model for the SFLP with general setup cost functions. The use of the general setup costs was motivated by an application in placing servers on the Internet. A company has to setup of number of servers to serve a number of clients. Since there is a connection cost between a server and a client, the costs of installing and maintaining the connection is a function of the amount of demand served by it. Because of economies of scale, these setup cost functions usually are concave. In order to solve the problem, a greedy algorithm was formulated.

Ghiani et al. (2002) discussed a Facility Location Problem in which several identical facilities can be opened at one site. They concluded that the standard solutions methods for CFLP fail to find a good solution. Their computational experiments had shown that the deviation of the solution provided by traditional Lagrange heuristics over the best lower bound can be as great as 500%. Therefore, they reformulated the problem as a Mixed Integer Program and developed an efficient Lagrangian heuristic algorithm. The results show that these heuristics perform well.

Wu et al. (2006) studied the CFLP with general setup cost functions. In their problem, it was also possible to locate several facilities at one site. The site setup cost was a fixed cost associated with opening a site and was independent of the facilities located in it. The

facility setup cost was a function of its size which is defined as the number of customers served by it in the uncapacitated problem or the number of facilities located in the site in the capacitated problem. To solve the problem, a Lagrange heuristic was used. These heuristics gave very good approximations of the solution for their large scale problems.

In this thesis, an adjusted CPLP model is used to solve the SCP. However, there are major differences in the costs of serving customers. Normally, these are made up by the transport costs from a facility to a customer. In the SCP, these costs are linked with the applications that are installed on a cluster. Also, a multicommodity variant of the CFLP problem is considered. Organizations can request a subset of a finite set of commodities, but a cluster has to serve all the demand of the assigned organizations. Up till now, there is no close example in the literature addressing this problem in the same way. In the following chapter, the proposed model will be discussed in greater detail.

2.3 Tabu Search

In the following articles a Tabu Search heuristic was used to achieve a solution for the Facility Location Problem.

Michel and Hentenryck (2004) investigated a Tabu Search heuristic for the *Uncapacitated Warehouse Location Problem* (UWLP). The neighborhood was extremely simply defined. It was obtained by just flipping the status of a warehouse; opened or closed. The algorithm used a Tabu-list which contained the set of warehouses that can not be flipped. At each iteration, the algorithm considered the set of neighbors which were not Tabu and the neighbor with the highest gain was selected. The used algorithm proved to be very robust and fast. In addition, it found optimal solutions (from problems of the OR library (Beasley, 1990)) very often. The variance on the quality of the solutions was always below 0.4% and zero most of the time.

Sun (2008) developed a Tabu Search heuristic procedure for the CFLP. It used a Move procedure to change the status of a facility. The neighborhood of a feasible solution was the set of distinct solutions that can be reached by making one move from the current solution. The procedure considered only feasible solutions, these were solutions where the capacity constraints were not violated. By fixing the status of a facility — opened or closed — the CFLP reduced to a Transportation Problem. A network algorithm was used to solve this problem. All solutions were stored, because solving these Transportation Problems took most of the computational time. A step in the algorithm was Tabu if it results in an already visited solution. Computational results on test problems from the literature and on test problems newly generated showed that this TS heuristic procedure

was very effective and efficient in finding good solutions. It found optimal solutions for almost all test problems in the literature.

Rolland et al. (1996) provided a Tabu search procedure for the p-median FLP. In this p-median problem exact p facilities had to be opened. The initial solution started with p opened facilities. The status of a facility can not easily be changed. By opening a facility another facility had to be closed, otherwise the solution was not feasible. This was done by a Nodes Substitution Procedure, a kind of paired Add and Drop procedure. First an additional facility was opened and total costs were calculated, then an opened facility was closed to obtain a feasible solution. Whether a facility was opened, the facility can not be closed for some steps. Closing a recently opened facility was Tabu. The obtained results were better than the results of a traditional Add/Drop or Swap procedure.

2.4 Genetic Algorithm

The use of Genetic Algorithm (GA) is very common in solving Facility Locations Problems. Next are a few recent articles discussing some of the findings in this field.

Kratica et al. (2001) considered a *Simple Plant Location Problem* (SPLP) and used GA to solve the problem. The algorithm they developed solved the SPLP with more than 1000 facility sites and customers. They found out that the techniques of rank-based selection and uniform crossover provided the best results. They also speed up the computational time by caching already computed object values. Considering their very large instances with more than 1000 facility sites and customers, GA produced qualitative solutions in reasonably short run time.

Chaudhry et al. (2003) addressed the uncapacitated p-median problem with maximum distance constraint. They used the normal techniques like mutation and crossover. However, a direct encoding method was adopted to represent the solution. Their algorithm was tested on two data sets, where the optimal solution was known beforehand. GA performed in both cases quite good compared to other solution methods. This finding was in contrary to earlier research, who reported non-promising results for this problem, but was in line with more recent research.

Correa et al. (2004) also proposed a GA for the p-median problem, but this time for the capacitated variant. Their GA also used direct encoding, where a gene consisted the opened facilities. The genetic operators were specifically developed for the p-median problem. In particular, they used a heuristic hyper-mutation operator, in addition with the crossover and conventional mutation operators. The algorithm was tested on a large scale and a real-world problem with a quite large search space, containing roughly 421

billion candidate solutions. They compared the results of their two GAs, one with hyper-mutation and one without it, with a Tabu Search heuristic. TS outperformed GA without the heuristic hyper-mutation operator; but the other GA outperformed TS. These results were evidence for the cost-effectiveness of the proposed heuristic operator, since all three algorithms assessed roughly the same number of candidate solutions during their search.

2.5 Stochastic Programming

An extension of the CFPL model is to consider some stochastic elements. In stochastic models some parameters in the objective or in the constraints are uncertain. The uncertainty is in most cases characterized by a probability distribution on the parameters. In most cases, only a few scenarios (possible outcomes of the data) or very specific joint probability distributions are specified. An example of few scenarios is that years can be good, fair, or bad, resulting in above average, average and below average yields of the different kinds of crops (Birge and Louveaux, 1997).

Stochastic models require one decision made at the beginning, so that the expected costs of this decision is minimized. Some popular stochastic models are the two-staged recourse models. Suppose a vector x the set of decision variables which are fixed in the first stage, when only partial information is available. In the second stage, all information is available and the second-stage variable y is minimized. Let ξ represent the realization of some random vector, q the recourse cost vector; W and T constraints matrices and requirement vector h when the full information is made available; and let A, c, and b the same as in the first stage. Finally, P is the non-negativity or integrality of the variables x and y. The stochastic program can be written as:

$$\begin{aligned} \min \quad & c^{'}x + \mathrm{E}_{\xi} Q\left(x, \xi\right) && (2.1)\\ \text{s.t.} \quad & Ax \leq b && (2.2)\\ & x \in P && (2.3) \end{aligned}$$

$$\begin{aligned} \text{where} \quad Q\left(x, \xi\right) = \quad \min \quad & q^{'}y && (2.4)\\ \text{s.t.} \quad & Tx + Wy \geq h && (2.5)\\ & y \in P && (2.6) \end{aligned}$$

$Q(x,\xi)$ is the optimal value of the second stage, given the scenario $\xi = (q,T,W,h)$ and the first-stage variables x. The expectation is taken with respect to ξ. In the first-stage the cost function (2.1) is minimized plus the expected costs of the second-stage decision. In the second-stage a recourse action can be taken where the term Wy compensates for a possible inconsistency of the system $Tx \geq h$. The costs of this recourse action are (2.4).

A popular adjustment of the two-stage recourse model is a model where the second stage is characterized by a finite set of scenarios. In scenario t, the constraint matrix, cost vector and requirement vector take on values T^t, W^t, q^t and h^t respectively, and scenario t occurs with probability p_t. It is now possible to write model (2.1)-(2.3) in extensive form as follows, where y^t represents the choice if scenario t materializes:

$$\begin{aligned}
\min \quad & c^{'}x + \sum_{t \in T} p_t \left(q^t\right)^{'} y^t && && (2.7)\\
\text{s.t.} \quad & Ax \leq b && && (2.8)\\
& T^t x + W^t y^t \geq h^t && t \in T && (2.9)\\
& \left(x, y^t\right) \in P && t \in T && (2.10)
\end{aligned}$$

Solving two-stage recourse models appears to be very hard. In model (2.7)-(2.10) the advantage is that risk is taken explicitly and it is a large scale LP model. As a consequence, a disadvantage is that if the number of scenarios is high, the model becomes too large to solve with commercial available LP/IP solvers like CPLEX. This is because the number of constraints and variables grow linear with the number of scenarios to power of number of independent random variables. However many heuristics exists to find solutions. Examples of solution methods include the L-shaped method, inner linearization methods and basis factorization methods. In addition, Non Linear Programming approaches exists like regularized decomposition, methods based on the stochastic program Lagrangian and the piecewise quadratic form of the L-shaped method. All these techniques are behind the scope of this thesis.

A method of reducing the number of scenarios in a stochastic program is to use Monte Carlo simulation (Shapiro and Philpott, 2007). In some programs the total number of scenarios is very large or infinite. If it possible to generate a idd (independent identically distributed) sample $\xi^1, \ldots, \xi^N$ of size N, $\mathrm{E}_\xi Q(x,\xi)$ can be approximated by the average;

$$\hat{q}_N(x) = N^{-1} \sum_{j=1}^{N} Q\left(x, \xi^j\right) \tag{2.11}$$

and the "true" value of model (2.1)-(2.3) by;

$$\min_{x \in X} \left\{ \hat{g}_N(x) := c^{'} x + N^{-1} \sum_{j=1}^{N} Q\left(x, \xi^j\right) \right\} \tag{2.12}$$

This technique is widely known under the name *sample average approximation* (SAA). A major question is in SAA is how large sample size N must be in order to achieve an accurate approximation of the true model. Since Monte Carlo simulation is used, there is possibility of a very slow convergence and in order to improve the accuracy N needs to increase substantially. Also, if the number of random variables is high the problems can not be solved with a high accuracy. The advantage of using Monte Carlo simulation is that it does not rely on the number of scenarios. It is proved and verified in problems with a manageable sample size, that the SAA method finds a solution of the original problem with a reasonable accuracy of 1% provided four conditions discussed in Shapiro and Philpott (2007).

An example of a stochastic two-stage *Stochastic Facility Location Problem* (SFLP) is given by Ravi and Sinha (2006). Ravi and Sinha considered an Uncapacitated Facility Location Problem where the demand of each client is not known in the first stage.

They first examined some properties of the model. For example, if the costs in the second-stage are identical to the first-stage it is possible to decouple the problem and solve each scenario separately. Secondly, if there is no second-stage and all the facilities had to be opened in the first-stage, the model boils down to a standard UFLP with some probability multipliers in the expected service costs. In this case existing approximations for UFL can be applied directly. The added difficulty, and indeed the interesting aspect of the model, arises from varying second-stage facility costs under different scenarios.

The main problem in the two-stage recourse model is that the different scenarios interact with the first-stage. The heuristic Ravi and Sinha proposed compares the solution obtained if all the demand is satisfied in the first-stage and the situation where no facilities are opened in the first-stage. This heuristics works in some cases, but it fails when there is interaction between scenarios. Their adjusted heuristics performs better in this case.

3 SaaS Clustering Problem

3.1 IP Formulation

The goal of the SCP is to obtain the optimal allocation of organizations $i \in I$ to clusters $j \in J$ in the SaaS network. The SaaS network consists of a set of clusters $l \in L$ where all the organizations are assigned to. The capacity a cluster q_l, $l \in L$ is related to the number of installed nodes $j \in J$. The higher the number of nodes in a cluster, the higher the capacity is. Each organization $i \in I$ requests a subset of all the available applications $k \in K$. In this subset L_i, the demand d_{ik} is defined as the amount of users in organization i·transactions per second for application k. Furthermore, the costs of installing application c_k, $k \in K$ only have to be paid once also when two organization on the same cluster requests the same program. In order to obtain an optimal allocation, the total costs of the SaaS network are minimized.

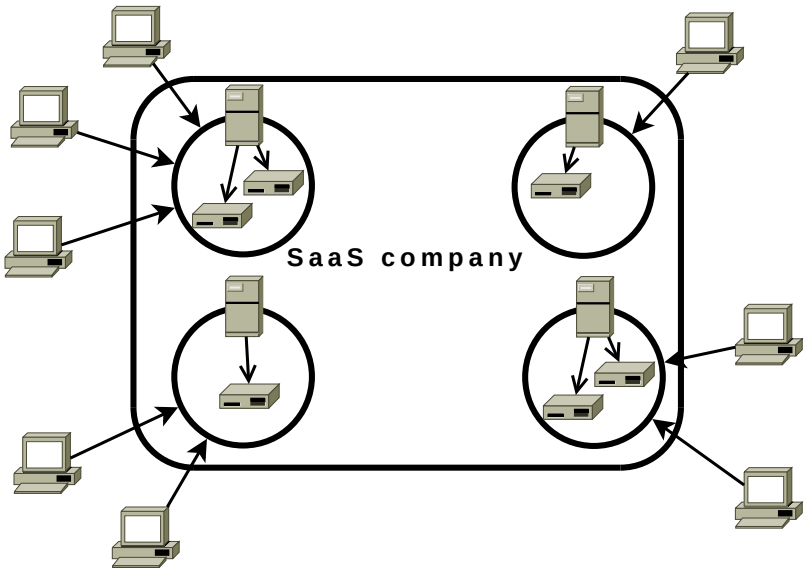


Figure 3.1: Concept behind the SCP

The idea behind the SCP is illustrated in figure 3.1. In this figure an organization is represented by a computer. An organization is connected to one cluster (server). A

cluster is collection of several nodes (racks). In the cluster all the needed applications to serve the connected organizations are installed. In addition, more than one organization can be connected to the same cluster.

To calculate the minimal costs the SCP is formulated as an adjusted Facility Location Problem. The IP formulation of the SCP is as follows:

$$\min \quad \sum_{k\in K}\sum_{j\in J} c_k z_{kj} + \sum_{j\in J} f y_j + \sum_{j\in J}\sum_{l\in L} e_l v_{jl} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ij} = 1 \qquad i \in I \tag{3.2}$$

$$\sum_{i\in I}\sum_{k\in L_i} d_{ik} x_{ij} \leq \sum_{l\in L} q_l v_{jl} \qquad j \in J \tag{3.3}$$

$$x_{ij} \leq y_j \qquad i \in I,\, j \in J \tag{3.4}$$

$$x_{ij} \leq z_{kj} \qquad k \in L_i,\, i \in I,\, j \in J \tag{3.5}$$

$$\sum_{l\in L} v_{jl} \leq 1 \qquad j \in J \tag{3.6}$$

$$x_{ij}, y_j, z_{kj}, v_{jl} \in \mathbb{B} \qquad i \in I,\, j \in J,\, k \in K,\, l \in L \tag{3.7}$$

c_k: The costs of installing application k.

e_l: The costs of opening l nodes.

d_{ik}: The demand of organization i for application k, defined as the amount of users·transactions per second.

x_{ij}: $\begin{cases} 1 & \text{if organisation } i \text{ is assigned to cluster } j. \\ 0 & \text{otherwise.} \end{cases}$

y_j: $\begin{cases} 1 & \text{if cluster } j \text{ is opened.} \\ 0 & \text{otherwise.} \end{cases}$

z_{kj}: $\begin{cases} 1 & \text{if application } k \text{ is installed on cluster } j. \\ 0 & \text{otherwise.} \end{cases}$

v_{jl}: $\begin{cases} 1 & \text{if in cluster } j,\, l \text{ nodes are opened.} \\ 0 & \text{otherwise.} \end{cases}$

i: Set of organizations.

j: Set of clusters.

k: Set of applications.

l: Set of nodes.

The object function (3.1) minimizes the total costs, which consist of the costs of placing applications in a cluster, setting up a cluster and installing nodes in a cluster. Constraints (3.2) define that for each organization all needed applications are located in one cluster. Constraints (3.3) state that a cluster can not serve more than the total capacity of the installed nodes. An important remark is here that there is a maximum amount of nodes in a cluster. It acts somewhere between the capacity constraint in a CFLP and that in the CFLP with multiple facilities on the same site, as described in (Ghiani et al., 2002). Constraints (3.4) ensure that an organization can only be served by a cluster that is opened. Constraints (3.5) check for every cluster which applications are placed on that cluster. It follows that an application only once has to installed in a cluster, if requested by multiple organizations. Constraints (3.6) are added to the model to ensure that only one set of nodes in a cluster is opened. Finally, (3.7) are the integrality constraints.

3.2 Lagrangian heuristic

A Lagrangian relaxation can be used to solve the model in section 3.1. The Lagrangian relaxation has been successfully used in various optimization problems as well as in *Capacitated Facility Location Problem* (CFLP) computation. See Fisher (2004); Wolsey (1998); Geoffrion and Bride (1978) for some examples.

The basis idea behind Lagrangian heuristics is to place hard constraints in the object function together with a Lagrange multiplier λ for every restriction. The fundamental step of the heuristic is to determine a lower bound. By means of the lower bound an upper bound can be found. If these bounds coincide, an optimal solution has been found. If not, the set of multipliers can be updated with use of a classical *subgradient algorithm*, which tries to find the highest lower bound as possible. At every step of the subgradient algorithm, a feasible solution of the original problem can be constructed, which can be used as a upper bound. The availability of both lower and upper bounds on the optimal objective function value is an attractive feature of the Lagrangian heuristic research. The relative gap between upper and lower bounds is typically used as a measure of the maximal error of the solution.

3.2.1 The lower bound estimation

A method to approach the solution of the SCP is to use a Lagrangian heuristic, based on a Lagrangian relaxation of the model (3.2)-(3.7). The only sensible restriction to relax is the demand constraint (3.2). Relaxing other constraints (except for (3.3), see subsection 3.2.2) does not make the Lagrange relaxation easier to solve.

Relaxing the demand constraints (3.2) in a Lagrangian fashion with multipliers λ_i, $i \in I$:

$$\begin{aligned} (\mathrm{LR}(\lambda)) \qquad \min \quad & \sum_{k\in K}\sum_{j\in J} c_k z_{kj} + \sum_{j\in J} f y_j + \sum_{j\in J}\sum_{l\in L} e_l v_{jl} \\ & + \sum_{i\in I} \lambda_i \left(1 - \sum_{j\in J} x_{ij}\right) \\ \text{s.t.} \quad & (3.3), (3.4), (3.5), (3.6), (3.7) \end{aligned} \tag{3.8}$$

For a given set of Lagrangian multipliers λ_i, $i \in I$, the optimal solution of $\mathrm{LR}(\lambda)$ can be found as follows. The model can be decomposed in $|J|$ sub-problems, one for each cluster j:

$$\begin{aligned} (\mathrm{LR}(\lambda) - j) \qquad \min \quad & \sum_{k\in K} c_k z_{kj} + \sum_{l\in L} e_l v_{jl} \\ & - \sum_{i\in I} \lambda_i x_{ij} && && (3.9) \\ \text{s.t.} \quad & \sum_{i\in I}\sum_{k\in L_i} d_{ik} x_{ij} \le \sum_{l\in L} q_l v_{jl} && j \in J && (3.10) \\ & x_{ij} \le z_{kj} && k \in L_i,\, i \in I,\, j \in J && (3.11) \\ & x_{ij}, z_{kj}, v_{jl} \in \mathbb{B} && j \in J,\, k \in K,\, l \in L && (3.12) \end{aligned}$$

Finally, the optimal solution of $\mathrm{LR}(\lambda) - j$ is given by $z^j_{\mathrm{LR}_\lambda} := \min\left\{f + z^j_{\mathrm{LR}(\lambda)}, 0\right\}$, $j \in J$ where $z^j_{\mathrm{LR}(\lambda)}$ is the optimal value of model (3.9)-(3.12). Remark that all sub-problems are equal, due to c_k. Model (3.9)-(3.12) is solved by using the Integer Programming solver of CPLEX. Then, the optimal solution value of $\mathrm{LR}(\lambda)$ is:

$$z^*_{\mathrm{LR}_\lambda} = \sum_{j=1}^{J} z^j_{\mathrm{LR}_\lambda} + \sum_{i\in I} \lambda_i$$

The solution $z^*_{\mathrm{LR}_{\lambda^k}}$ of the Lagrangian relaxation LR(λ) provides a lower bound to the original problem, since

$$z^*_{\mathrm{LR}_{\lambda^k}} \leq z_{IP}$$

That this relation hold can be proved by noting that the feasible region of the original IP formulation is smaller than the feasible region of the Lagrange relaxation.

3.2.2 Alternative lower bound estimation

Another Lagrangian relaxation of the model (3.2)-(3.7), is to relax capacity constraint (3.3).

Relaxing the capacity constraints (3.3) in a Lagrangian fashion with multipliers $\lambda_j \geq 0$, $i \in I$:

$$\begin{aligned}
(\mathrm{LR}(\lambda)) \qquad \min \quad & \sum_{k\in K}\sum_{j\in J} c_k z_{kj} + \sum_{j\in J} f y_j + \sum_{j\in J}\sum_{l\in L} e_l v_{jl} \\
& + \sum_{j\in J} \lambda_j \left(\sum_{l\in L} q_l v_{jl} - \sum_{i\in I}\sum_{k\in L_i} d_{ik} x_{ij} \right) \\
\text{s.t.} \quad & (3.2), (3.4), (3.5), (3.6), (3.7)
\end{aligned} \tag{3.13}$$

Note that in this case model can not be further decomposed. For a given set of Lagrangian multipliers $\lambda_i \geq 0$, $i \in I$, the optimal solution of LR(λ) can be found as follows. Again, the model is solved by using the Integer Programming solver of CPLEX. The optimal solution value of LR(λ) is $z^*_{\mathrm{LR}_\lambda}$. The solution $z^*_{\mathrm{LR}_\lambda}$ of the Lagrangian relaxation LR(λ) provides also a lower bound to the original problem.

3.2.3 The upper bound estimation

By solving the Lagrangian relaxation, a set of opened cluster J^* is obtained and for every cluster the amount of opened nodes. To obtain an upper bound the following linear programming problem is used to find a feasible solution:

$$
\begin{aligned}
\text{(UR)} \qquad \min \quad & \sum_{j\in J^*}\sum_{k\in K} c_k z_{kj} && && (3.14)\\
\text{s.t.} \quad & \sum_{j\in J^*} x_{ij} = 1 && i\in I &\\
& \sum_{i\in I}\sum_{k\in L_i} d_{ik}x_{ij} \le \sum_{l\in L} q_l v^*_{jl} && j\in J^* && (3.15)\\
& x_{ij} \le z_{kj} && k\in L_i,\, i\in I,\, j\in J && (3.16)\\
& x_{ij}, z_{kj} \in \mathbb{B} && i\in I,\, j\in J^*,\, k\in K && (3.17)
\end{aligned}
$$

The optimal solution value of UR, denoted as $z^*_{\text{UR}} = z_{\text{UR}} + \sum_{l\in L} e_{jl}v^*_{jl} + \sum_{j\in J} f_j y^*_j$ is an upper bound on for model (3.2)-(3.7).

However, it may not always be the case that the opened amount of clusters y_j and nodes v^*_{jl} are feasible for the original model. Therefore, a greedy algorithm is used in order to satisfy all the demand before running model (3.14)-(3.17). This algorithm adds per iteration a node with the lowest costs, until all demand is satisfied.

Step 1: Calculate the cost of adding an extra node δ_j for every cluster $j \in J^*$;

$$
\delta_j = \begin{cases} f + e_1 & \text{if } y^*_j = 0 \\ e_{l+1} & \text{if } y^*_j = 1 \end{cases}
$$

Step 2: Add an extra node — where it is possible — to the cluster $j \in J^*$ with the lowest costs δ_j.

Step 3: If all demand can be satisfied; STOP, otherwise go to step 1.

An other upper bound can be obtained by using later discussed rules of thumb. An advantage of these rules is the very fast calculation time, but there is no guarantee that is upper bound is good.

3.2.4 The Lagrangian Heuristic

The proposed Lagrangian heuristic follows the next steps (if the lower bound of subsection (3.2.1) is chosen):

Step 0: Initialize LB $= -\infty$, UB $= \infty$ or set UB equal to the solution of the rules of thumb, $m = 0$, $k = 1$, $\lambda^k_i = 0$ or some other specified value like $\min(c_k)$,

$i \in I$. Select a tolerance value $\varepsilon \geq 0$ and the maximum number $maxiter > 0$ of subgradient iterations since LB latest increase.

Step 1: Solve LR $\left(\lambda^k\right) - j$. If $z^*_{\mathrm{LR}_{\lambda^k}} > \mathrm{LB}$, then set $\mathrm{LB} = z^*_{\mathrm{LR}_{\lambda^k}}$ and $m = 0$; else set $m = m + 1$.

Step 2: Optional: Solve UR $\left(\lambda^k\right) - j$. If $z^*_{\mathrm{UR}_{\lambda^k}} < \mathrm{UB}$, then set $\mathrm{UB} = z^*_{\mathrm{UR}_{\lambda^k}}$.

Step 3: If $\max\left(\lambda^{k-1} - \lambda^k\right) \leq \varepsilon$, then STOP, LB and UB correspond respectively to the best lower and upper bound on z^*. In particular, if LB = UB, the optimal solution is found.

Step 4: If $m = maxiter$, then $\alpha = \alpha/2$ and set $m = 0$.

Step 5: Determine the sub gradient of each relaxed constraint:

$$s_i^k = \sum_{j \in J} x_{ij}^k - 1, \qquad i \in I \tag{3.18}$$

where x_{ij}^k, $i \in I$, $j \in J$, corresponds to the optimal solutions of LR_{λ^k}. Set:

$$\lambda_i^{k+1} = \lambda_i^k + \beta^k s_i^k, \qquad i \in I \tag{3.19}$$

where β^k is computed according to:

$$\beta^k = \frac{\alpha\left(\mathrm{UB} - z^*_{\mathrm{LR}_{\lambda^k}}\right)}{\sum_{i \in I}\left(s_i^k\right)^2} \tag{3.20}$$

Set $k = k + 1$ and return to step 1.

Formulas (3.18)-(3.20) update the Lagrangian multipliers according to the classical sub gradient method. In particular, (3.18) defines the sub gradients as the violations of the relaxed constraints (3.2); (3.19) calculates the new multipliers; (3.20) computes the step sizes in (3.19). Note that in (3.20) the parameter α is used to limit the variation on the values of the Lagrangian multipliers when the lower bound is approaching the upper bound. With the aim to improve the performance of the heuristic, α is halved every *maxiter* iterations of last change in the lower bound. In case of the other lower bound of subsection (3.2.2), the Lagrange heuristics follows the same step, but in Step 5 the the

sub gradient is defined in a different way. The sub gradient is defined as follows:

$$s_j^k = \sum_{i \in I} \sum_{k \in L_i} d_{ik} x_{ij} - \sum_{l \in L} q_l v_{jl}, \qquad j \in J \tag{3.21}$$

The rest of the changes should be clear for the reader.

Due to the assumption of homogeneous servers, the costs c_k of installing application $k \in K$ are the same for every server. As a result, the first Lagrangian relaxation of subsection 3.2.1 can not distinguish among different servers and assigns the same organizations to every cluster. In lower bound many organization will not be served and are penalized by the Lagrange multipliers. Therefore, the Lagrangian heuristic will maybe give bad bounds.

The second Lagrange relaxation of subsection (3.2.2) does not have this problem, but in this case the wisest action would be to install all the programs on only one cluster. Since the capacity restriction does not have to hold, the total needed capacity of this cluster is very high. The Lagrange relaxation will open empty clusters with no organization assigned to it in order fill up the remaining demand. As a result, this lower bound will be very close to the LP bound of model (3.2)-(3.7).

3.3 Local Search Techniques

3.3.1 Tabu Search

A Tabu Search heuristic is used to achieve a feasible solution in reasonable time. Tabu Search (TS) is a local search technique proposed by Glover in the 70's (Glover and Laguna, 1997). The basic principle of Tabu Search is to pursue a local search whenever it encounters a local optimum by allowing non-improving moves. Cycling back to previously visited solutions is prevented by the use of a memory, called the Tabu list. This list records the recent history of the search. The basic Tabu Search can be seen as a combination of a local search with short-term memory (Gendreau, 2003). This subsection will give a general description of TS and a detailed description of the TS used in this thesis.

The search space of the TS heuristic is the space of all possible solutions that can be considered during the search. The heuristic starts with an initial feasible solution. This initial solution can be an arbitrary feasible solution or the outcome of another heuristic, for example a Genetic Algorithm or some rules of thumb. This solution is stored as the best solution.

In every step of the heuristic the neighborhood structure of the current solution is considered. These neighborhoods can be reached by a single local transformation. The best neighbor in the neighborhood is chosen. In a normal local search this is the neighbor with the most positive gain with respect to the previous solution. In TS the neighbor can have a worse objective value with respect to the previous solution. In this way TS avoids the heuristic to stop in a local optimum. If the current solution is better than the best solution, the current solution is stored as the best solution.

Tabu lists (Tabus) are used to prevent cycling when moving away from local optima through non-improving moves. Normally in the next step the heuristic can return in the local optimum. But now some steps are Tabu. The moves that reverse the effect of recent moves are not allowed. Tabus are stored in a short-term memory of the search (the Tabu list). The Tabus are in fact recently visited solution, but storing whole solutions is not very efficient. Therefore, the most commonly used Tabus involve recording the last few transformations performed on the current solution for prohibiting reverse transformations.

Sometimes Tabus are too powerful: they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process. It is thus necessary to use algorithmic devices that will allow one to cancel Tabus. These are called aspiration criteria. The simplest and most commonly used aspiration criterion consists in allowing a move, even if it is Tabu, if it results in a solution with an objective value better than that of the current best-known solution.

A normal local search stops when no better solution can be achieved. However, in the Tabu Search the solution in the next step can be a worse solution than the current. In theory, the search could go on forever, unless the optimal value of the problem is known. Therefore a termination criteria has to be chosen, for example: stop after a fixed number of iterations, stop after a fixed amount of time, stop after some fixed number of iterations without an improvement in the best solution or stop when the objective value reaches a pre-specified threshold value.

In every step all neighborhood solutions has to be considered. This can take a long time if the data set becomes large. An alternative is to consider only a random sample of the neighborhoods. This reduces the computation time, but very good solutions can be missed.

The Tabu Search can be improved in several ways, for example intensification, diversification and allowing infeasible solutions. Intensification means that the steps taken in the Tabu Search are investigated. Steps that take place very often will be done in the initial solution. Diversification means that the heuristic starts with a new different

initial solution, to explore another part of the search space. Allowing infeasible solutions means that infeasible solutions are allowed by relaxing the constraints. If a constraint is violated, this costs a penalty.

In the next steps the TS used in this research is described in detail. These steps follows the algorithmic description of TS described in (Zhang and Sun, 2002). Let S_b denote the best solution obtained so far and TL the Tabu list.

Step 1: Initialize: Generate an initial solution x. Let $S_b = x$, $k = 1$, $TL = \Phi$.

Initial solution x is obtained by placing every organization at a separate cluster:

$$x_{ij} = 1 \qquad \forall i = j \tag{3.22}$$

Set $k = 1$, $TL = \Phi$, $length(TL) = 8$

Step 2: Generate candidates: Consider the solutions in the neighborhood of x, which are obtained by a single local transformation of x. Randomly pick out a certain number or pick all solutions in this neighborhood to form the candidate set $N(x)$.

Two local transformations of x are considered: Merge/Divide and Swap. A Merge/Divide transformation means that an organization is placed on a different cluster. Organizations can so be merged at a cluster, or can be divided if there is more than one organization on a cluster:

$$x_{ij} = x_{ij'} \qquad \forall i, j \neq j' \tag{3.23}$$

Costs can differ through a Merge/Divide transformation, for the configuration of the clusters change. The total number of applications to be installed can decrease by merging organizations to a cluster. If the demanded applications of the merged organizations (partly) overlap each other, the objective value decreases, while the applications has to be installed only one time in a cluster ($z_{kj} \in \mathbb{B}$). The first part of the objective function, $\sum_{k \in K} \sum_{j \in J} c_k z_{kj}$, decreases in this way. The number of opened cluster can also change. By merging organizations to a cluster, clusters can become empty (no organization is dedicated to the cluster) and the empty cluster can be closed. The second part of the objective function, $\sum_{j \in J} f y_j$, decreases in this way. The third part of the objective function, $\sum_{j \in J} \sum_{l \in L} e_l v_{jl}$, can decrease, for the number of needed nodes can decrease by merging organizations. The objective function consists of three parts, and there is a kind of trade-off between these three parts. Therefore, a Merge transformation is not always beneficial and even a Divide transformation, which can open an additional cluster, could be beneficial.

A Swap transformation means that two organizations, which are placed on different clusters, are inter-changed:

$$x_{ij} = x_{ij'} \qquad x_{i'j'} = x_{i'j} \qquad \forall i \neq i', j \neq j' \tag{3.24}$$

In a Swap transformation the number of opened cluster does not change, but the number of installed applications and needed nodes can change. By executing these two local transformation, infeasible neighbor solutions can be obtained. A neighbor can be infeasible due to the maximum amount of 20 nodes in a cluster. This is not allowed, and these neighbors are therefore not considered. Only a randomly chosen subset of the neighborhood is considered, because considering all neighborhoods can take much computation time. The organizations with a bad overlap of applications in their cluster get a higher probability to be selected for considering. The weight of an organization to be selected is defined as:

$$weight_{i'} = 1 - \frac{\sum_k 1_{((\sum_{i,j} d_{ik} x_{ij} - d_{i'k}) > 0)}}{\sum_k 1_{(d_{i'k} > 0)}} \tag{3.25}$$

$weight_i = 1$ if no applications of organization i are already in cluster j. $weight_i = 0$ if all applications of organization i are already in cluster j. The probability to be selected is not the same as the weight of an organization, for the sum of all weight can be more than 1. The probability is defined as:

$$prob_i = \frac{weight_i}{\sum_i weight_i} \tag{3.26}$$

50% of the organizations is randomly chosen with the calculated probabilities, for it is reasonable to think that it is beneficial to merge organizations on a cluster with overlapping applications.

Step 3: Move: Find out the best solution $y \in N(x)$. If $y \in TL$ and $S_y < S_b$ (no better overall solution) let $N(x) = N(x) - \{y\}$ and repeat step 3 with the adjusted $N(x)$. Else, let $x = y$ and let $S_b = y$ if y is better than S_b.

The local transformation which results in the best improvement in objective value of current solution x is executed, if it is not Tabu. If the best solution y is Tabu, but better than S_b the local transformation also is executed. y is set to x and if y is better than S_b, then $S_b = y$.

Step 4: Output: If termination condition is satisfied, stop and output S_b. Otherwise, let $TL = TL \cup \{x\}$ and remove first element of TL. Let $k = k + 1$ and go back to

step 2.

As termination criteria a maximum number of steps k is set to 200. In every step not the visited solution is set in the Tabu List, but the inverse of the executed local transformation. This means that in a Merge/Divide transformation the organization can not go back to the cluster where it comes from and in a Swap transformation the two inter-changed organizations can not be inter-changed again. In addition a Tabu is added to prevent combination-steps. These are consecutive steps of the Merge/Divide transformation. They occur if two organizations, i and i' are placed on cluster j. If organization i is placed by the heuristic at cluster j', where there are still no other organizations, then in the next step of the heuristic organization i' can be merged again with organization i on cluster j'. The heuristic will in this way not improve the objective value, therefore it is Tabu to execute a similar local transformation where an organization goes from cluster j to cluster j' on another organization for the length of the Tabu list.

The whole procedure is executed several times with different initial solutions. All initial solutions are obtained by the procedure described in step 1, but now 25 random merges (if they result in a feasible solution) are done for diversification of the initial solution. The procedure is also executed with the results of the rules of thumb, described in subsection 5.2.5.

3.3.2 Genetic Algorithm

Genetic Algorithm (GA) was first described in the book: *Adaptation in Natural and Artificial Systems* (Holland, 1975). The book of Holland created a new field of research and applications that goes much wider now than the original GA he proposed. Nowadays, the terms Evolutionary Computing or Evolutionary Algorithms (EAs) are used when talking about GA. However, in the context of meta-heuristics, GAs in their original form describe most of the main ideas.

While Holland's influence in the development of GA has been very significant — several other scientists with different backgrounds also developed similar ideas. In 1970s Germany, Rechenberg (1973) and Schwefel (1977) developed the idea of the Evolutionsstrategie (in English, evolution strategy), while — in the 1960s — Bremermann (1958), Fogel et al. (1966) and others in the USA implemented their idea for what they called evolutionary programming.

All these ideas are centered around the principles of mutation and selection, which lay at the core of the neo-Darwinian theory of evolution. Although at the beginning some promising results were obtained, GA did not really take off until the 1980s. Not the least

important reason for this was that the techniques needed a great deal of computational power.

Throughout GA, solutions that are also called individuals take part in a reproductive process in which they interact, mix together and produce offspring that retain the good characteristics of their parents. The reproductive process which involves the creation of new solutions is based on selection, crossover, inheritance and mutation.

The first question in GA is the size of the initial population, and secondly the method in choosing individuals. The size of the population should be intuitively be considerable big to allow variation in the search place, but a very large population means a loss in efficiency. It would take too much computation time to find a solution. A reasonable population size for the test cases in section 5.2 was 100. The initialization of the population is chosen to be random, like most GA applications (Reeves, 2003).

The basic idea of selecting an individual $h \in H$ for a next generation is related with the fitness. The fitness values are calculated by the following formula:

$$fit_h = \sum_{k \in K} \sum_{j \in J} c_k z_{kj} + \sum_{j \in J} f y_j + \sum_{j \in J} \sum_{l \in L} e_l v_{jl} \tag{3.27}$$

For explanation of the parameters c_k, z_{kj}, f, y_j, e_l and v_{jl} see subsection 3.1. The goal of GA is to minimize the total costs, which consist of the costs of placing applications in a cluster, setting up a cluster and installing nodes in a cluster. In every generation, several individuals are drawn from the population with the use of a probability distribution and will then be placed in the next generation. The chance of selecting an individual is based on formula 3.35. The original scheme for selecting individuals is commonly known as the roulette-wheel method (Mitchell, 1998). It uses a probability distribution for selection in which the selection probability of a given string is proportional to its fitness. By using a random number generator it is possible to generate a set of chosen individuals. The GA for the SCP model uses the roulette-wheel method for selection. However, other ways of selection are possible — for example — stochastic universal selection and tournament selection (Reeves, 2003).

In most of the GAs binary encoding is used to represent a solution. However, this is not well suited for the SCP model. Therefore, a direct encoding method is adopted where the location of the organizations are represented as follows:

$$S = \{s_1, s_2, \ldots, s_i\} \tag{3.28}$$

where the allele s_i denotes the assigned cluster of organization i, $s_i \in J$ and $J =$

$\{j_1, j_2, \ldots, j_n\}$. With S the values of x_{ij}, y_j, z_{kj} and v_{jl} can be determined. A cluster $j \in J$ is opened ($y_j = 1$), if $j \in S$, otherwise $y_j = 0$; the cluster that serves organization $i \in I$ is defined as $x_{ij=s_i} = 1$, elsewhere $x_{ij \neq s_i} = 0$; the installed nodes v_{jl} can be obtained by the minimal amount of nodes necessary to serve all the organizations for every cluster $j \in J$ and the installed applications on cluster $j \in J$ $z_{kj} := 1_{\{\sum_{i \in I} d_{ik} x_{ij=s_i} > 0\}}$.

Mutation involves a probability that an arbitrary allele in a genetic sequence will be changed from its original state. In a mutation a randomly chosen organization $i \in I$ in cluster j is added to other cluster $j^* \neq j$ if possible. The most used method is to generate a random variable for each allele. However, a more sophisticated method can be used. For every allele in the genetic sequence the current overlap on cluster $j^* \in J$ in programs ξ_i for every organizations $i \in I$ is determined, where ξ_i is defined as:

$$\xi_i = \frac{\sum_{k \in L_i} w_{kj^*}}{\sum_{k \in L_i} z_{kj^*}} \tag{3.29}$$

where $w_{kj^*} = 1$ if application k has overlap on its current cluster j^*, otherwise $w_{kj^*} = 0$. The chance that a particular allele mutated uses the roulette-wheel method with a chance of selecting $1/\xi_i$. Suppose that $j = \{1, 2, \ldots, 6\}$ and $i = \{1, 2, \ldots, 6\}$, thus an individual consist of six alleles, i.e.

$$M : (1, 5, 6, 3, 5, 2) \tag{3.30}$$

The roulette-wheel chooses — for example — the third allele. The location of this allele j^* will be mutated and it is possible that a new cluster will be opened for this allele. For all the possible mutations the new overlap with the programs on the new cluster $\phi_{j'}$ is calculated. $\phi_{j'}$ is defined as follows:

$$\phi_{j'} = \frac{\sum_{k \in L_i} w'_{kj'}}{\sum_{k \in L_i} z_{kj'}} \quad j' \neq j^* \tag{3.31}$$

where $w'_{kj'} = 1$ if application k has overlap on the new cluster j', otherwise $w'_{kj'} = 0$. The higher the new overlap is, the better the mutation is. The roulette-wheel method is used to pick the mutation which is carried out. This results in a new individual M':

$$M' : (1, 5, 3, 3, 5, 2) \tag{3.32}$$

The purpose of mutation in GAs is to allow the algorithm to avoid local minima by preventing the population from becoming too similar to each other, thus slowing or even stopping the algorithm. Note that in some cases a mutation leads to an unfeasible

solution. In this case the mutation is not carried out.

An extension of the mutation method is the so called hybrid mutation. In a hybrid mutation all organizations on a cluster get mutated in the same way. Instead of calculating the individual overlap for every organization $i \in I$, the mean overlap per cluster $j \in J$ is calculated. The higher this number, the higher the probability that the cluster is chosen. The newly combined cluster for the hybrid mutated alleles follows the same approach as a normal mutation.

A crossover is simply a matter of replacing some of the characteristics in one parent by the corresponding characteristics of the other. In a crossover some organizations — currently in the cluster — are replaced by organizations of another cluster. Two parents P_1 and P_2 are first randomly selected from the population. As before suppose, i.e.

$$P_1 : (1, 5, 6, 3, 5, 2) \quad \text{and} \quad P_2 : (1, 4, 6, 2, 1, 2) \tag{3.33}$$

which represent two possible solutions to a problem. A crossover swaps one variable from P_1 to P_2. One swap point is chosen in a similar way as formula 3.29. A new solution is produced by combining the pieces of the the original 'parents'. For instance, if the swap point was the fourth allele, the offspring solutions would be

$$C_1 : (1, 5, 6, 2, 5, 2) \quad \text{and} \quad C_2 : (1, 4, 6, 3, 1, 2) \tag{3.34}$$

In its initial approach by Holland (1975) GA used selection, recombination and mutation to form the new generation. This way looks from an optimization point of view a strange thing to do — a considerable effort is spend to obtain a good solution, only to run the risk of throwing it away and thus preventing it from taking part in further reproduction. For this reason, de Jong (1975) introduced the concepts of elitism and population overlaps. His ideas are simple — an elitist strategy ensures the survival of the best individual(s) so far by preserving it and replacing only the remaining $(H - 1)$ members of the population with new individuals. Overlapping populations take this a stage further by replacing only a fraction G (the generation gap) of the population at each generation. Only elitism is used in the GA for the SCP problem.

In contrary to simple neighborhood search methods that stop after reaching a local optimum, GAs can in theory run for ever. To avoid this, a termination criterion is needed. The most simple way is to stop after predetermined number of generations. Otherwise include, stopping after a fixed number of fitness evaluations or running time. On the other hand, the algorithm can aborted when the diversity falls below some threshold.

For the last case, it is in most cases not really clear how to track diversity (Reeves, 2003). In the GA for the SCP the heuristics stops after a certain number of generations, which was defined at the beginning of the algorithm.

Algorithm 3.1 gives the pseudo code of the GA used to solve the SCP problem. The results of the algorithm are presented in section 5.2.

3.3.3 Link between TS and GA

There are some similarities between the techniques used in TS and GA. A Merge/Divide transformation in TS can be compared with a mutation and a hybrid in GA. A mutation assigns an organization to another cluster. This is also done by Merge/Divide in TS. A Hybrid combines two clusters into one. This can also be done by the Merge/Divide in TS, but in a different way. It occurs if an organization which is separately assigned to a cluster j is assigned to another cluster j'. Organizations are no longer assigned to cluster j, and cluster j is closed. By closing a cluster by a Merge/Divide transformation, two clusters are combined into one. A Swap transformation can be compared with a Crossover. Here, two organizations are interchanged form cluster. Figure 3.1 gives an overview of the similarities between the techniques used in TS and GA.

	TS	GA
Technique 1	Merge / Divide	Mutation Hybrids
Technique 2	Swap	Crossover

Table 3.1: Overview techniques

There are also differences between TS and GA. The main difference with the TS algorithm is that GA does not explore the neighborhood of a single solution but perform a search in the neighborhood of a population of solutions. In addition, TS performs one local transformation in every step of the heuristic, but GA performs multiple transformation in every individual in each heuristic-step. An important difference is also that TS does not work with random transformations, in contrast to GA.

Algorithm 3.1 Pseudo code GA

Parameters:

nGen: The amount of generations.

n: The amount of individuals per generation.

PercMut: Percentage of individuals that will mutate.

PercHyb: Percentage of individuals that will hybrid.

PercCross: Percentage individuals that will crossover.

Variables:

Pop: Collection of individuals (current generation)

PopNew: Collection of individuals (new generation)

Pseudo code:

- Initialize Pop with population of size n.
- Calculate cost f_i for every combination i in Pop.
- Calculate nMerge = PercMerge·n, nHyb = PercHyb·n and nCross = PercCross·n.
- For 1:nGen
  - PopNew = $\emptyset$.
  - Add n-nMut-nCross individuals from Pop to PopNew; with the chance of selecting a individual:

$$\frac{\exp\left(fit_i - fit_{min}\right)}{\sum_i \exp\left(fit_i - fit_{min}\right)} \tag{3.35}$$

  - Choose nMut of individuals from Pop and mutate if possible.
  - Choose nHyb of individuals from Pop and hybrid if possible.
  - Choose nCross of individuals from Pop and crossover if possible.
  - Add the mutated and crossover individuals to PopNew.
  - Pop = PopNew.
  - Calculate cost f_i for every combination i in Pop.
  - Store the best value, with best combination.
  - Print the best individual.

4 Stochastic SaaS Clustering Problem

4.1 Stochastic IP-formulation

An extension of the SCP is to consider a second-stage. Some assumptions were made; there are a finite number of scenarios $t \in T$ which can occur in the second-stage with probability p_t, $t \in T$. Also, in this stage the opened cluster y_j, $j \in J$ are the same, but additional nodes can be purchased and added to cluster only if the cluster was already opened. In the second-stage the demand of an organization is given by d^t_{ik}, $i \in I$, $k \in K$, $t \in T$. The demand of organization $i \in I$ can change in respect to the first-stage in two ways. First, the demand in terms of transactions can change for the already requested applications. Second, the list of requested programs L^t_i, $i \in I$, $t \in T$ can be different in scenario t; the organization can request some new applications or stop using some applications. There is a possibility that the total required capacity in cluster $j \in J$ is larger then total capacity q_l, $l \in L$ in the second-stage; if this occurs a penalty cost pen_{tj}, $t \in T$, $j \in J$ have to be paid in order to compensate for violating the capacity limit. In addition, in the second-stage it is not possible to resell nodes that are currently not used anymore. Thus, the number of nodes v^t_{jl}, $j \in J$, $l \in L$, $t \in T$ is always equal or greater the number of nodes v^0_{jl}, $j \in J$, $l \in L$ in the first-stage. The place of an organization x_{ij}, $i \in I$, $j \in J$ is the same in both stages, meaning that is not possible to reallocate an organization in the second-stage. Moreover, if an application is installed in the first-stage and is also requested in the second-stage, maintenance costs g^t_k, $k \in K$, $t \in T$ have to be paid. If an application is installed in the second-stage, installation costs z^t_{kj}, $k \in K$, $j \in J$, $t \in T$ have to be paid, these installation cost can differ from the costs in the first-stage. The goal is to minimize the total costs of the first-stage plus the recourse costs in the second stage.

$$\min \quad \sum_{k\in K}\sum_{j\in J} c_k^0 z_{kj}^0 + \sum_{j\in J} f y_j + \sum_{j\in J}\sum_{l\in L} e_l^0 v_{jl}^0 + \sum_{t\in T} p_t \left(\sum_{k\notin L_i^0, k\in L_i^t} \sum_{j\in J} c_k^t z_{kj}^t + \right. \tag{4.1}$$

$$\left. \sum_{k\in L_i^0, k\in L_i^t} \sum_{l\in L} g_k^t z_{kj}^t + \sum_{j\in J}\sum_{l\in L} e_l^t v_{jl}^t - e_l^t v_{jl}^0 + \sum_{j\in J} pen_{tj} \right)$$

s.t.

$$\sum_{j\in J} x_{ij} = 1 \qquad i \in I \tag{4.2}$$

$$\sum_{i\in I}\sum_{k\in L_i^0} d_{ik}^0 x_{ij} \leq \sum_{l\in L} q_l v_{jl}^0 \qquad j \in J \tag{4.3}$$

$$\sum_{i\in I}\sum_{k\in L_i^t} d_{ik}^t x_{ij} \leq \sum_{l\in L} q_l v_{jl}^t + pen_{tj} \qquad j \in J,\, t \in T \tag{4.4}$$

$$\sum_{l\in L} q_l v_{jl}^t \geq \sum_{l\in L} q_l v_{jl}^0 \qquad j \in J,\, t \in T \tag{4.5}$$

$$x_{ij} \leq y_j \qquad i \in I,\, j \in J \tag{4.6}$$

$$x_{ij} \leq z_{kj}^0 \qquad k \in L_i^0,\, i \in I,\, j \in J \tag{4.7}$$

$$x_{ij} \leq z_{kj}^t \qquad k \in L_i^t,\, i \in I,\, j \in J,\, t \in T \tag{4.8}$$

$$v_{jl}^0 \leq 1 \qquad j \in L,\, l \in L \tag{4.9}$$

$$v_{jl}^t \leq 1 \qquad j \in L,\, l \in L,\, t \in T \tag{4.10}$$

$$x_{ij}, y_j, z_{kj}^0, z_{kj}^t, v_{jl}^0, v_{jl}^t \in \mathbb{B} \qquad i \in I,\, j \in J,\, k \in K, l \in L,\, t \in T \tag{4.11}$$

c_k^0: The costs of placing application k in the first stage.

c_k^t: The costs of placing application k in scenario t.

z_{kj}^0: $\begin{cases} 1 & \text{if application } k \text{ is installed on cluster } j \text{ in the first stage.} \\ 0 & \text{otherwise.} \end{cases}$

z_{kj}^t: $\begin{cases} 1 & \text{if application } k \text{ is installed on cluster } j \text{ in scenario } t. \\ 0 & \text{otherwise.} \end{cases}$

v_{jl}^0: $\begin{cases} 1 & \text{if in cluster} j\, l \text{ nodes are opened in the first stage.} \\ 0 & \text{otherwise.} \end{cases}$

v_{jl}^t: $\begin{cases} 1 & \text{if in cluster} j\, l \,\text{nodes are opened in scenario}\, t. \\ 0 & \text{otherwise.} \end{cases}$

g_k^t: The costs of maintaining application k in scenario t.

p_t: The probability of scenario t.

d_{ik}^0: The demand of organization i for application k in the first stage, defined as the amount of users·transactions per second in the first stage.

d_{ik}^t: The demand of organization i for application k in scenario t, defined as the amount of users·transactions per second in scenario t.

pen_{tj}: Penalty term for violating the capacity constraint in scenario t for cluster j.

t: Set of scenarios.

The object function (4.1) minimizes the total costs, which consist of the costs of installing applications in a cluster, setting up a cluster and installing nodes in a cluster in the first stage and the mean total costs in the second stage, which consists of the cost of installing a new application in a cluster, the maintaining costs of the already installed programs in the first stage, the costs of buying additional nodes and the penalty term for violating the capacity constraints. Constraints (4.2) define that for each organization all needed applications are located in one cluster. Constraints (4.3) state that a cluster in the first stage cannot serve more than the total capacity of the installed nodes. In constraints (4.4) the needed nodes for the scenarios in the second stage are set. It is possible that the capacity of the cluster is not sufficient to serve the new total demand of the assigned organizations in the second-stage, if this occurs a penalty term has to be paid in order to compensate for violating the capacity limit. Constraints (4.5) check that the nodes opened in a cluster in every scenario is at least the number of nodes in the first stage. Constraints (4.6) ensure that an organization can only be served by a cluster that is opened. Constraints (4.7) check for every cluster which applications are placed on that cluster in the first stage; in the second-stage constraints (4.8) also check which applications are in the cluster. Constraint (4.9) and (4.10) state that in both the first- and second-stage only one set of nodes can be opened in a cluster. Finally, (4.11) are the integrality constraints for all the variables.

4.2 Local Search Techniques

4.2.1 Stochastic Tabu Search

In order to solve the stochastic SCP both local search techniques — Tabu Search and Genetic Algorithm — were adjusted. TS, as described in section 3.3.1, will stay roughly the same, but the following changes and assumptions were made.

In the original SCP, the SaaS company had to made a decision for only one period and knew all the demand from the organizations. Now, there is a second period — the second-stage — were the demand for applications of an organization is uncertain. The SaaS company must decide in the first-stage how to cluster the organizations, in order to minimize the total costs. An organization can in theory request all the possible combinations of applications and the demand in terms of requested transactions per second per application can be every positive number. This results in an infinite number of scenarios in the second-stage.

Since it is practically impossible to consider all the scenarios because of the computational time, the assumption is made that an organization can request every application, but if it requests an application the demand (d_{ik}^t) is fixed in every scenario. This reduces the number of scenarios to $2^{\#\,organizations \cdot \#\,applications}$. However, since the number of organizations and applications can be very large, the number of scenarios is still too large to consider.

Inspired by the method of *Sample Average Approximation* (SAA) the following idea is developed to obtain a set of scenarios which will hopefully describe the situation in the second stage well. The demand d_{ik}^t is determined and set fixed in every scenario. After that, a number of randomly generated scenarios are drawn. In a scenario an organization will request a certain number of applications. By selecting a large number of scenarios and evaluating the costs; the scenarios would describe the second-stage well because of SAA.

The costs or benefits of a local transformation in TS is in the SSCP determined by relation (4.1). This means that the local transformation has to be evaluated in all the scenarios for the second stage. This costs a lot of time, and therefore a trade-off has to be made between the number of scenarios and the running time of TS.

Since it is in theory possible for an organization to request all the combinations of applications in the second-stage, it is not useful to transform the selecting procedure given by relation (3.25) and (3.26). If all possible scenarios are considered, this will not influence the probability. Therefore, the probability for an organization to be selected depends only on the overlap in the first stage.

4.2.2 Stochastic Genetic Algorithm

The idea of SAA can also be incorporated in GA. The assumption that the demand (d_{ik}^t) is fixed and known for the second-stage is made, but in the first-stage which applications an organizations request is not known. The idea is generate a large number of scenarios to obtain the expected costs for the second-stage. Due to SAA these expected costs will be very close to the real costs.

In order for GA to work with SAA, the fitness function for individual $h \in H$ is changed;

$$fit_h = \sum_{k \in K} \sum_{j \in J} c_k z_{kj} + \sum_{j \in J} f y_j + \sum_{j \in J} \sum_{l \in L} e_l v_{jl} + \mathrm{E}_\xi Q(x, \xi) \tag{4.12}$$

where $\mathrm{E}_\xi Q(x, \xi)$ the expected costs for the second-stage are. These costs are calculated with SAA. Also, in the reproductive processes there are some changes. When calculating the current overlap of an individual $h \in H$, the expected overlap of the second-stage are as well added. The results of both stochastic local search techniques are in section 5.3.

5 Computational results

5.1 Randomly generated data

5.1.1 Deterministic SCP

In this study eight data sets were used to examine the performance of the IP model and the heuristics for the deterministic SCP. In these data sets a distinction is made among different types of organizations, namely large, moderate and small organizations. The characteristics of these organizations are shown in table 5.1. The organizations differ in the number of applications needed. A large organization requests more applications than a smaller organization. There is also a distinction in the number of transactions needed. Large organizations request more transactions than smaller organizations. These parameters are uniformly distributed on the stated interval, but the requested applications can only be integer and the requested transactions can only be thousands.

The parameters for the capacity of the nodes, the costs of the nodes, the maximum amount of the nodes and the amount of applications are the same in every data set. It is assumed that the capacity of the nodes in a server has a concave form. This means that adding a node to a server does not increase the server capacity with the capacity of the node, but with a fraction of the node capacity. This fraction decreases with respect to the amount of nodes in a cluster. This relation is shown in formula 5.1;

$$\frac{10000}{1 + 0.05 \cdot (k-1)} \quad \text{with } k = \#\text{ nodes in a cluster} \tag{5.1}$$

The costs of opening a cluster is fixed at €1000 and the costs of adding nodes is

	Demand (applications)	Demand (transactions)
Large organization	10 - 12	8000 - 10000
Moderate organization	7 - 9	5000 - 7000
Small organization	4 - 6	2000 - 4000

Table 5.1: Characteristics of organizations

Type of application	Costs	# Applications
Common-use	80 - 120	10
Large	160 - 200	10
Moderate	110 - 150	10
Small	60 - 100	10
Rest	60 - 200	20

Table 5.2: Characteristic of applications

linear in the amount of nodes. Adding a node costs €500. The maximum amount of nodes in a cluster is set to 20. This is due to the size of a rack enclosure (±40U) divided by the size of a rack (2U). The number of applications is 60. The applications are divided in five application groups, namely common use, large, moderate, small and rest. This is shown in table 5.2. Every organization uses 1/3 of its applications from the common use programs. Another 1/3 comes from the type specific applications. The large organizations use large and therefore expensive programs, moderate organizations use moderate and moderate-priced applications and small organizations use small and cheap applications. The last 1/3 of the demanded applications is randomly chosen from the rest programs and the other type specific applications. For a large organization this means that this 1/3 is chosen from the rest, moderate and small programs. The rest programs have a four times higher probability to be chosen than the other type-specific applications. The installation costs (c_k) are also shown in table 5.2. The costs is a ten and are determined by an integer value drawn from an uniform distribution on the stated interval.

With these parameters eight data sets were created. The data sets differ in the combination of different types of organizations. Four data sets were created with 60 organizations and another four data sets were created with 120 organizations. The details are shown in table 5.3. Another remarks is that all costs are divided by ten, to have some smaller numbers in the data sets.

5.1.2 Stochastic SCP

The data sets described in subsection 5.1.1 were also used in the stochastic SCP for the first-stage. The demand in the second-stage d^t_{ik} was determined by the following procedure. First, d_{ik} was determined with the parameters shown in table 5.1. As described in section 2.1 the expected growth of the SaaS market will be 20% per year. Therefore a random multiplier μ_{ik}, which is uniformly distributed on the domain $[0.9 : 1.2]$ is drawn

	# Large	# Moderate	# Small	# Total
Data set 1	20	20	20	60
Data set 2	40	40	40	120
Data set 3	40	10	10	60
Data set 4	10	40	10	60
Data set 5	10	10	40	60
Data set 6	30	60	30	120
Data set 7	30	30	60	120
Data set 8	60	30	30	120

Table 5.3: Organization composition of the data sets

and multiplied with d_{ik}. This procedure was executed for every i and k and the results hold in every scenario. In this way, variable demand was obtained for the second stage.

The second step was to select the applications which an organization will request. The number of requested applications was the same as shown in table 5.1. But here also a random multiplier μ_{ik}, uniformly distributed on $[0.9 : 1.2]$ were drawn and multiplied with the number of requested applications. The results were rounded and requested applications were chosen in the same way as described in subsection 5.1.1.

With this procedure 50 scenarios are obtained, which will hopefully describe the situation in the second-stage well.

The stochastic SCP considers two-stages. The installation costs in the first-stage c_k^0 was the same as c_k in table 5.2. The installation costs in the second-stage c_k^t was determined by $c_k + 30$, for it is assumed that installing a new application was more expensive in the second-stage than in the first-stage. The maintenance costs g_k^t were defined as $c_k - 40$, because maintaining applications seems to be cheaper than installing the application.

5.2 Deterministic part

All results reported in this section were obtained on a Core2 Duo PC with 3.0 GHz and 2048 Mbytes of RAM. The LP/IP formulations were programmed in GAMS 23.0 and with the LP/IP solver of CPLEX 11.0.1. Both heuristics — TS and GA — were programmed in MATLAB R2007b.

		IP solver	TS	GA	LP bound	Dif LP/IP
Case 1	Solution	1594	1596	1594	1510.41	5.24%
	Time (s)	4	51	81	1	
Case 2	Solution	2197	2197	2197	2034.97	7.38%
	Time (s)	47	50	90	1	
Case 3	Solution	1611	1615	1623	1528.71	5.11%
	Time (s)	4	51	82	1	
Case 4	Solution	1258	1258	1274	1132.23	10.00%
	Time (s)	24	51	87	1	

Table 5.4: Results Small Test case

5.2.1 Justification heuristics

To provide a justification for the use of TS and GA four small test cases were made. These small test case were subsets of test case 1, as described in subsection 5.1.1. The number of clusters, organizations, nodes and applications were reduced to 12. The number of organization types in the test cases is respectively (4,4,4), (8,2,2), (2,8,2) and (2,2,8). Table 5.4 shows the obtained results. The IP formulation gives all the four cases proved optimal solutions. TS found the optimal solution twice, which was also the case in GA. In the other cases where TS or GA did not find the optimal solution, the results were pretty close. The time it took to find the solution is the lowest for directly solving the IP formulation with CPLEX, while running GA took the most time. The quite long running time of TS with respect to directly solving the IP formulation can be explained by two things. First the number of iterations in the heuristic is set to 200. Second through diversification the heuristic is run ten times with different initial solution. So running TS only ones takes about 5 seconds. All IP solutions are about 5 - 10% away from the LP bound. This indicates that the LP lowerbound can be quite far from the IP solution.

TS and GA found the (almost) optimal solution in all of the small test cases and this justifies the later use of the heuristics in larger data sets. The IP model then could not always be solved due to computational limits, but TS and GA will most likely give a pretty good approximation of the best solution.

5.2.2 Results IP model

The results of the IP model are shown in table 5.5. None of the eight cases could be solved optimal in a reasonable time. CPLEX found a solution for the small cases 1, 3,

	IP-model				
	Time	Solution	LB	Gap	Div
Case 1	10:00	29969	19078.51	36.34%	31.54%
Case 2	30:00	-	38209.45	-	-
Case 3	10:00	40214	27678.25	31.17%	19.90%
Case 4	10:00	24004	18012.56	24.96%	11.96%
Case 5	10:00	16378	12332.13	24.70%	12.05%
Case 6	30:00	-	37184.07	-	-
Case 7	30:00	-	31025.98	-	-
Case 8	30:00	-	45053.04	-	-

Table 5.5: Results IP-model

4 and 5, but after finding a feasible solution the solution was not improved anymore. The found integer solutions of the these cases were quite high with respect to the lower bound. These lower bounds were not or almost not improved during the computational process. As a result, the duality gap was in every case more than 20% and in some cases more than 30%. In the large cases 2/6/7/8, where there are 120 organizations (instead of 60 in the other cases), CPLEX did not found a feasible solution of these case. However, only a lower bound was given. The column Div represents the difference between the solution of the IP model and the best known solution. These best known solution were obtained by TS, which will be discussed later on. These differences are all bigger than 10% and in case 1 even more than 30%. Because of the bad bounds of the IP formulation and the large difference between the given solution and the best known solution it is concluded that solving the IP formulation directly is not useful to obtain a good solution in large problems.

5.2.3 Comparison lower bounds

In table 5.6 several different lower bounds for all the eight cases of the SCP are presented. The first lower bound was calculated by relaxing the integrality constraints 3.7 to real values. This LP bound can be calculated optimally very fast in the small cases 1/3/4/5. However, in the large cases 2/6/7/8, the time to calculate the optimal value is almost 18 times higher. These solutions were used to benchmark the Lagrange relaxations.

The Lagrange lower bounds based on subsection 3.2.1 are shown in the columns Lagrange 1. In all the results was $\alpha = 0.5$ and halved after 20 iterations of no improvements. In the all test cases a positive lower bound was found. Even the fear of bad bound stated in subsection 3.2.4 proved to be of no concern. Only in small test case the lower bound

	LP bound		Lagrange 1			Lagrange 2		
	Time (m)	Solution	Iter	Time (m)	Solution	Iter	Time (m)	Solution
Case 1	0:11	19078.51	400	39:04	21739.98	400	61:10	18858.32
Case 2	5:32	38209.45	400	213:38	33771.91	65	199:28	7237.00
Case 3	0:30	27678.25	400	48:46	32596.74	400	52:25	26675.62
Case 4	0:10	18012.56	400	53:36	18687.74	400	25:02	17916.61
Case 5	0:15	12332.13	400	30:33	13839.06	400	23:43	12323.29
Case 6	7:21	37184.07	200	20:29	13944.48	35	122:54	7059.00
Case 7	6:28	31025.98	200	27:22	15291.53	35	139:12	5990.00
Case 8	7:40	45053.04	200	24:10	25717.50	35	105:32	8425.00

Table 5.6: Deterministic SCP lower bounds

was higher then the LP bound. In the other cases the bound was lower, but this most likely due to α used in the subgradient and the number of no improvements which result in halving α and of course the number of iterations. More investigation is needed in order to find a better α. Then most likely all the bounds will be higher then the LB bound.

On the other hand, the second Lagrange lower bound based on subsection 3.2.2 performed in the way as suspected. In the small cases, the lower bound became very close to the LP bound, which was expected in subsection 3.2.4. But the biggest issue with this lower bound was the computational time. In the large cases it took three hours to calculate 65 iterations, while around 400 were needed to obtain a good lower bound. This lower bound will at most be equal to the LP bound, since in the best solutions of heuristics shared many similarities with the LB bound. Thus it can be concluded that the most efficient way to obtain a lower bound is the LP relaxation, but it is possible to obtain better bounds with the first Lagrange relaxation. The second relaxation proved to be not much use.

5.2.4 Local Search Techniques

Since the IP formulation did not obtain a good solution of the problem, Tabu Search and Genetic Algorithm were used to find a better feasible solution. The results of these two heuristics are shown in table 5.7. For all cases solutions were found. The running time of the two heuristics are comparable, but TS takes a bit more time than GA in general.

The results of TS were the best found solution in all cases, but the results of GA were pretty close. The column Div gives the difference between the two solutions. More-

	TS			GA			
	Time (s)	Solution	Gap	Time (s)	Solution	Div	Gap
Case 1	252	22784	16.26%	230	23170	1.69%	17.66%
Case 2	953	46320	17.51%	828	47305	2.13%	19.23%
Case 3	248	33541	17.48%	228	33689	0.44%	17.84%
Case 4	256	21439	15.98%	230	21802	1.96%	17.38%
Case 5	256	14617	15.63%	228	14941	2.22%	17.46%
Case 6	2215	44866	17.12%	1639	45954	2.24%	19.08%
Case 7	2349	37727	17.76%	1528	38061	0.89%	18.48%
Case 8	2303	54497	17.33%	1246	55127	1.16%	18.27%

Table 5.7: Deterministic Results

over, if compared with the results in table 5.5, both heuristic perform better with less computational time. However, since both heuristics are local search techniques no indication of how close the solutions are to the optimal value can be given, since the heuristics could be trapped in a local minimum. The results can only be compared with the LP lower bound from the previous subsection. In the column Gap, the relative gap ((solution − LP bound) /LP bound) is given. This percentage indicated how far the solution lays compared to the LP bound. Given the results for the small test cases in subsection 5.2.1, the gaps now were a little bit higher. Solving the large cases 2/6/7/8 took more or less four times the computational time of the small cases. This is due to the doubled number of organizations and clusters. Still, the computational time was reasonable and a solution was found, in contrast with table 5.5.

5.2.4.1 Results Tabu Search

The results of TS are shown in table 5.8, figure 5.1 and 5.2. Table 5.8 gives the amount of opened clusters and statistics about the number of nodes in a cluster. It can be seen that the mean number of opened nodes is quite high with respect to the maximum number of nodes, which is 20. In case 3 and 7 the mean number of opened nodes is the smallest. This is due to the relative large number of large organizations in this case. It is difficult to merge large organizations on a cluster, due to the capacity constraints. The last column, mean excess capacity, points out to the mean not used capacity in a cluster. This is the difference between the capacity of the opened nodes and the dedicated demand.

Figure 5.1 shows the amount of organization in a cluster. The colored bars indicate the total number of organizations and the distribution of organization-types. Figure 5.2 gives a graphical representation of the installed applications in a cluster. The colored bar

	# Clusters	# Nodes			
		Mean	Min	Max	Mean excess capacity
Case 1	18	18.67	16	20	1309.5
Case 2	36	19.11	16	20	1764.6
Case 3	31	15.71	11	20	2447.0
Case 4	17	18.65	10	20	1537.8
Case 5	11	19.36	18	20	1528.1
Case 6	36	18.56	10	20	1418.0
Case 7	31	17.91	11	20	1337.9
Case 8	44	18.50	11	20	1803.0

Table 5.8: TS Results

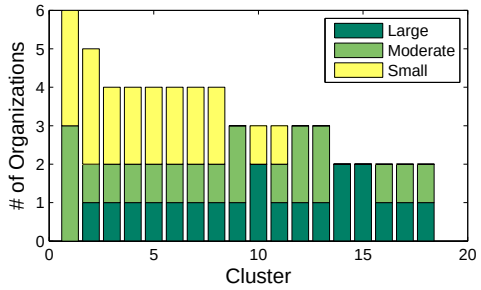
indicates the total number of applications installed in a cluster and the color indicates for how many organizations the applications is installed. In this way, the overlap of applications can be seen. In all cases, except for case 3, 30% - 40% of the applications installed on a cluster are requested by two or more organizations. In case 3, this number is lower due to the high ratio of large organizations.

5.2.4.2 Results Genetic Algorithm

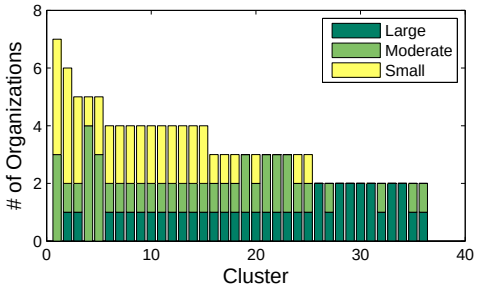
The GA results were obtained with an initial population size of 100 individuals. The amount of generations was 500. Finally, the percentage elitism was 40%; crossover 25%; mutation 25% and hybrid mutation 10%. The effects of these chosen parameters is later discussed in subsection 5.10.

Table 5.9, figure 5.3 and 5.4 show the results of GA. If compared with TS, GA opened more clusters in general. As a consequence, the mean number of nodes in a cluster was less than in TS. Also, the minimum amount of nodes opened in a case was also lower in most instances. But more important, the excess capacity was always bigger than in TS, except for case 2 and 8. This indicates that GA was in most cases less efficient than TS. Figure 5.3 shows the number of organizations per cluster and figure 5.4 gives the amount of installed applications in a cluster.

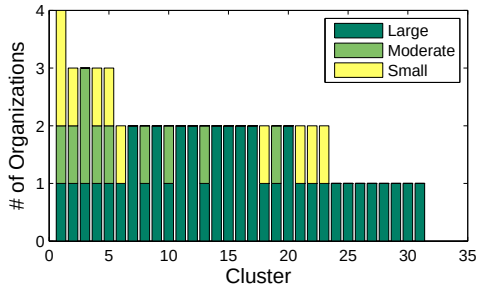
In figure 5.4 it can be seen that the overlap in the GA cases is around 30% - 20% of the applications, which lower then TS. This also indicates that GA was performing worse than TS, since it was more beneficial to assign organization with high overlap on the same cluster.



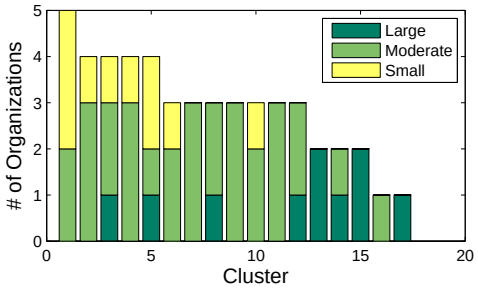
(a) Case 1



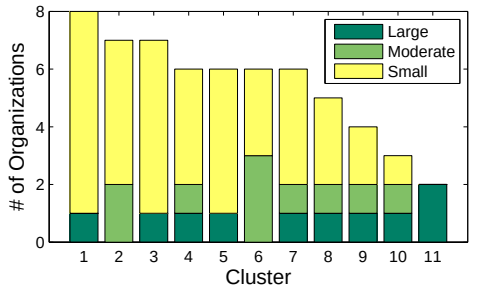
(b) Case 2



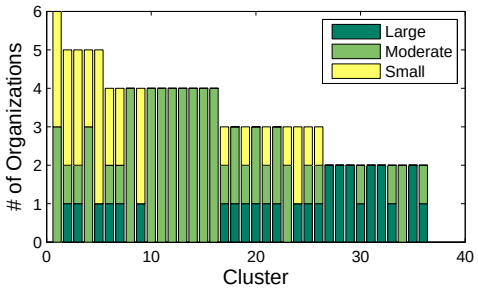
(c) Case 3



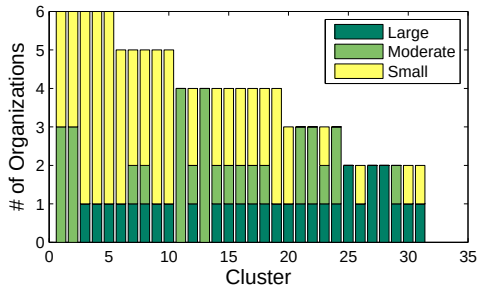
(d) Case 4



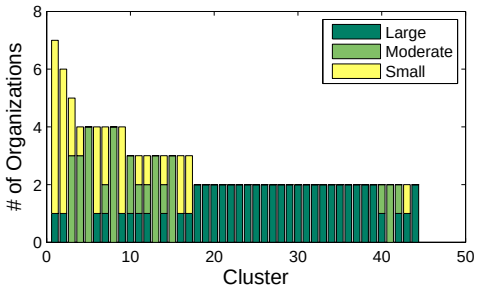
(e) Case 5



(f) Case 6

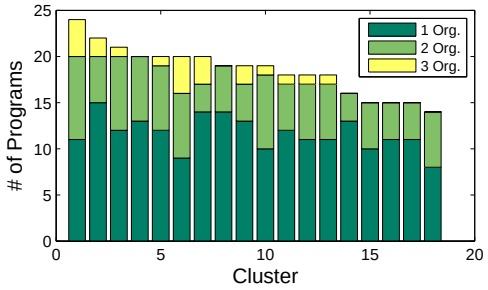


(g) Case 7

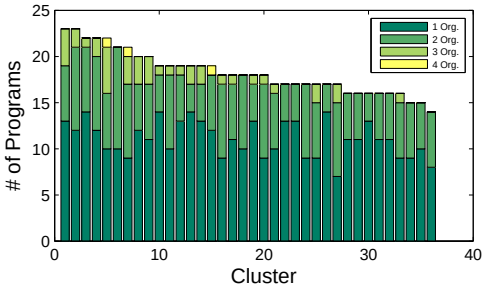


(h) Case 8

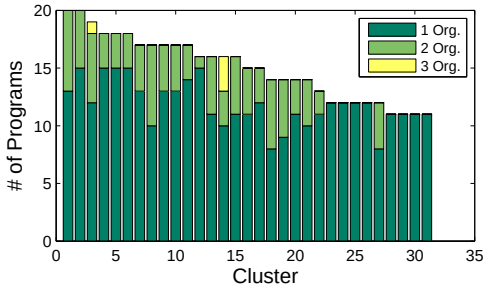
Figure 5.1: TS results: Number of organizations



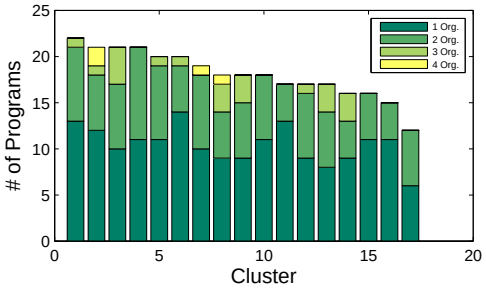
(a) Case 1



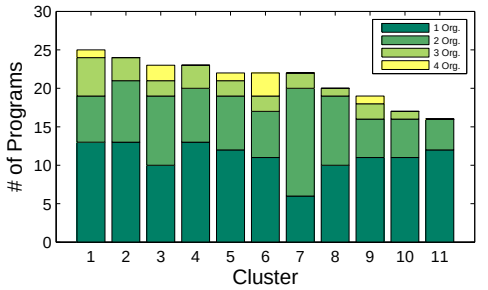
(b) Case 2



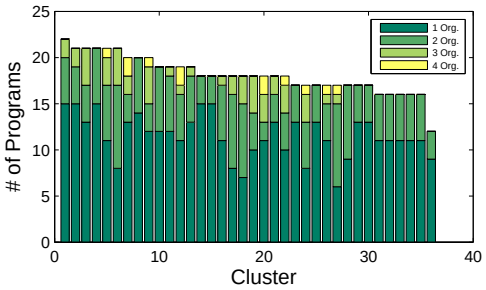
(c) Case 3



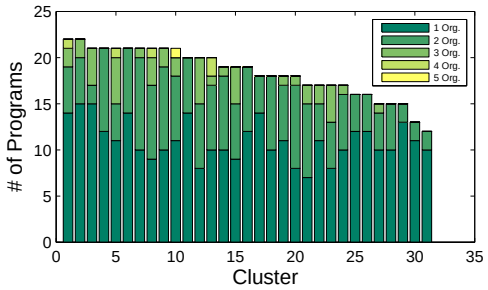
(d) Case 4



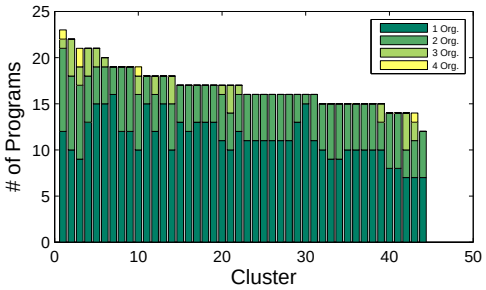
(e) Case 5



(f) Case 6

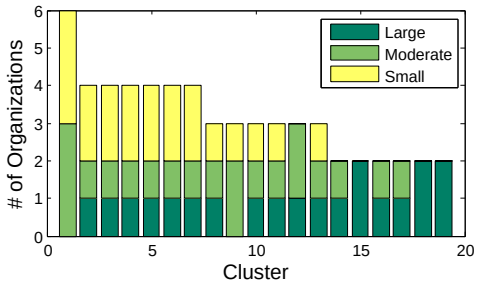


(g) Case 7

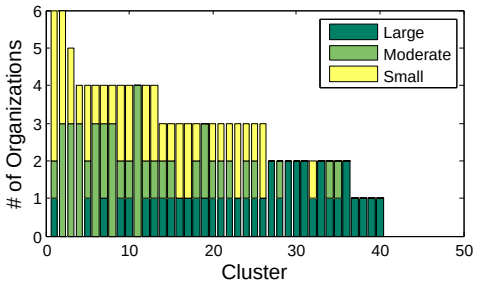


(h) Case 8

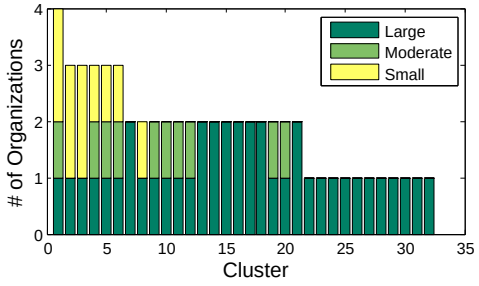
Figure 5.2: TS results: Number of applications



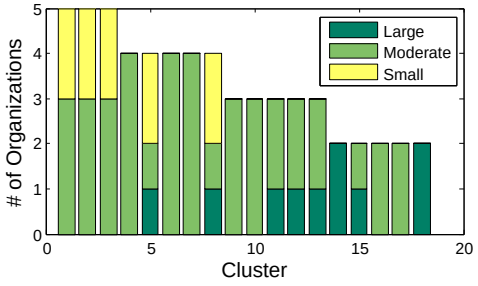
(a) Case 1



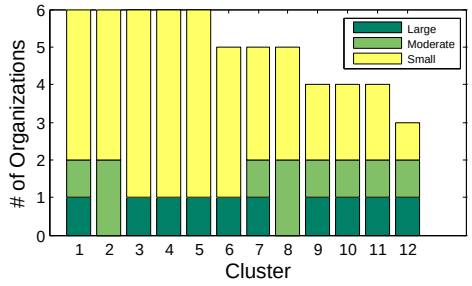
(b) Case 2



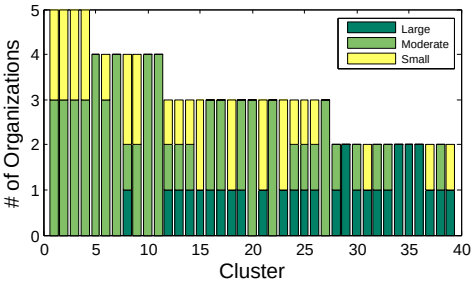
(c) Case 3



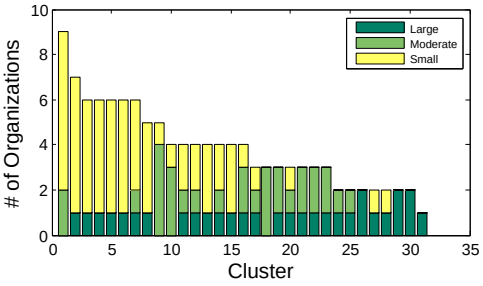
(d) Case 4



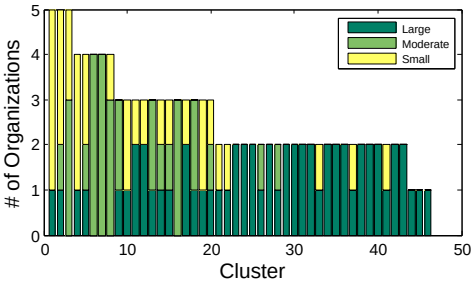
(e) Case 5



(f) Case 6

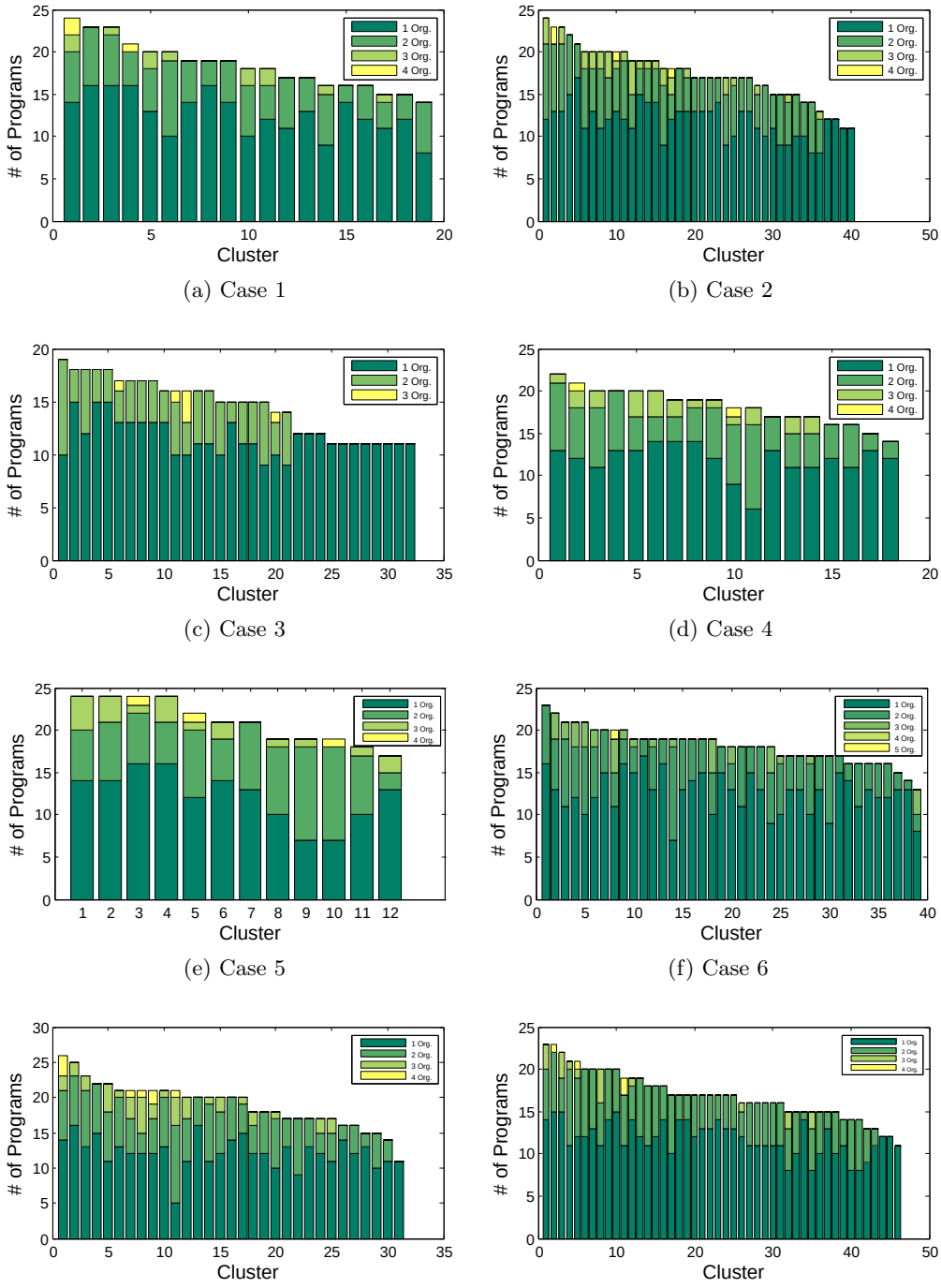


(g) Case 7



(h) Case 8

Figure 5.3: GA results: Number of organizations



(a) Case 1

(b) Case 2

(c) Case 3

(d) Case 4

(e) Case 5

(f) Case 6

(g) Case 7

(h) Case 8

Figure 5.4: GA results: Number of applications

	# Clusters	# Nodes			
		Mean	Min	Max	Mean excess capacity
Case 1	19	17.74	13	20	1766.9
Case 2	40	17.20	10	20	1635.7
Case 3	32	15.21	11	20	2385.4
Case 4	18	17.61	10	20	1478.8
Case 5	12	17.75	13	20	1440.5
Case 6	39	17.18	11	20	1833.9
Case 7	31	17.97	11	20	1952.4
Case 8	46	17.70	11	20	1745.3

Table 5.9: GA Results

5.2.5 Rules of thumb

Since TS gave the best solutions of the problems, some rules of thumb will be derived from the test case solutions to give guidelines for clustering in SaaS networks.

- Merging large organizations on a cluster is quite hard. Therefore, dedicate only one large company to a cluster.
- Try first to merge moderate organizations with large organizations and after that try to merge small organizations to the opened clusters.
- Try to use the maximum capacity in a cluster, setting up a new cluster is more costly.
- Merging organizations with overlapping applications is beneficial, however in general about 40% of the installed applications in a cluster is only demanded by one organization.

These rules of thumb can be derived of the obtained results. With these rules, clustering in SaaS networks is more intuitive. Table 5.10 gives the results of applying these guidelines.

By only applying the rules of thumb the results are between 3 till 8 percent from the best known solution (obtained by TS in section 5.2.4.1). The two cases with a high number of large organizations, namely case 3 and 8, have the worst results. This is due to the first guideline: dedicate only one large organization to a cluster. If the obtained solutions are used as initial solution for TS, the results become better and are about 1 percent away from the best known solution. By setting the obtained results as initial

	Results	Div	Results after TS	Div
Case 1	23618	3.66%	22989	0.90%
Case 2	47749	3.09%	46804	1.04%
Case 3	35523	5.91%	33768	0.68%
Case 4	22309	4.06%	21608	0.79%
Case 5	15262	4.41%	14577	-0.27%
Case 6	45939	2.39%	44931	0.14%
Case 7	38589	2.28%	37624	-0.27%
Case 8	58629	7.58%	55115	1.13%

Table 5.10: Results guidelines

solution for TS, the TS is intensified and diversified. Intensification means that the steps taken in TS are investigated and steps that take place very often will be done in the initial solution. Diversification means that the heuristic starts with a new different initial solution, to explore another part of the search space. The most remarkable results are obtained in case 5 and 7, where the solution is lower than the best known solution so far.

It follows that clustering by the rules of thumb give an acceptable result, but the results can improve by inserting these solution as initial solution in TS.

5.2.6 Parameter Analysis

It is important to investigate the influence of the parameters which are chosen in the local search techniques. These parameters can have a large influence on the search progress and the final solution. In this subsection therefore a parameter analysis will be carried out. This is done for case 1 and case 2. The main difference between these two cases is that case 2 has twice the amount of organizations compared to case 1.

5.2.6.1 Tabu Search

In TS two parameters were used in the search progress namely the number of steps in the algorithm k and the length of the Tabu List. In the analysis the Tabu List varies from 6 till 12 and k varies from 100 till 1000. To compare the effects, four Tabu Searches were carried out with different Tabu Lists lengths and at certain step k the best solution was given. Table 5.11 shows the results of the parameter analysis. The initial solutions were obtained by the rules of thumb from subsection 5.2.5 (23618 for case 1 and 47749 for case 2).

		Case 1			Case 2		
Length TL	k	R. time (s)	Solution	Diff.	R. time (s)	Solution	Diff.
6	100	13	23066	-2.34%	49	46877	-1.83%
	250	32	23059	-2.37%	120	46873	-1.83%
	500	62	22987	-2.67%	239	46784	-2.02%
	1000	153	22812	-3.41%	599	46784	-2.02%
8	100	12	23093	-2.22%	45	46900	-1.78%
	250	30	22917	-2.97%	112	46900	-1.78%
	500	60	22849	-3.26%	225	46744	-2.10%
	1000	151	22840	-3.29%	569	46723	-2.15%
10	100	13	23089	-2.24%	47	46666	-2.27%
	250	31	22918	-2.96%	116	46469	-2.68%
	500	62	22825	-3.36%	229	46396	-2.83%
	1000	152	22824	-3.36%	569	46339	-2.95%
12	100	12	22996	-2.63%	46	46846	-1.89%
	250	30	22996	-2.63%	116	46839	-1.91%
	500	60	22996	-2.63%	237	46839	-1.91%
	1000	152	22996	-2.63%	611	46839	-1.91%

Table 5.11: TS Parameter Analysis

The best result for case 1 was obtained with $k = 1000$ and $length(TL) = 6$ and $k = 1000$ and $length(TL) = 10$ in case 2. It is clear that the results are the best with $k = 1000$, for the best solution can only be improved by iterating. However with $length(TL) = 12$ the search progress will not improve the solution after $k = 100$ in case 1 and $k = 250$ in case 2. It is not easy to say what length of TL was optimal. There were some random effects in TS for selecting the organizations and the search progress can be highly influenced by these random effects. Therefore, it is not possible to say that the corresponding $length(TL)$ of the best solution is the optimal length of the Tabu List. Since it is 10 in case 1 and 6 in case 2, any Tabu List length between 6 and 10 might be sufficient. 12 or longer seems to be too long, since TS did not found better solutions after respectively 100 and 250 iterations.

5.2.6.2 Genetic Algorithm

GA depends on several parameters which all have an effect on the quality of the solution. The first parameter was the population size. The size of the population was very important; when the size is too low the variation of individuals would be very small. However, a very large population means longer computational time. A population of 50,

100 and 200 individuals was considered. The next set parameters were related with the reproductional features of the population. With elitism the best randomly chosen individuals were unchanged placed in the next generation. It was investigated whether this strategy is meaningful. The techniques crossover and mutation — which were discussed in subsection 3.3.2 — were set equal to each other. Also, the effects the stronger form of mutation — hybrids — were investigated. And finally, the different solutions of number of generation were compared.

In table 5.12 the results of different input parameters can be seen for case 1 and case 2. In case 1, the population with only 50 individuals found the lowest solution at 500 iteration. This result was not in line the reasoning given earlier. However, in case 2 the population with 200 gave the best solution. Having elitism or not, did not gave much different solutions. In case 1, no elitism gave lower costs; while in case 2 it was the other way around. The effects of having hybrids were more significant. In both cases, the solution was considerably lower if there were hybrids. Finally, the number of generation was also from great importance. Since GA in practice could be running forever, an iteration limit has to be set. Around 750 iterations, there was no real significant reduction in the total costs anymore. All the obtained results showed the solution found by the rules of thumb for both case 1 and case 2 could be improved.

5.3 Stochastic part

In this section the results for the Stochastic SCP are presented. Only the local search techniques — TS and GA — were executed and just on case 1 and 2. The IP formulation from section 4.1 was not implemented, for it is assumed not being solved in reasonable time. In the deterministic part, it was hard to find a feasible solution with CPLEX, so it will be even harder in the stochastic part. Only case 1 and 2 were investigated due to the computational time of the heuristics.

Table 5.13 shows the outcomes of TS and GA. TS gave the best results in both cases. This is in line with the findings in the deterministic part. The difference between the solutions is again given in the Div column. In case 1, the computational time is comparable, but in case 2 the time to execute the TS was much larger than the time of GA. This is due to the great number of costs evaluations in TS. However, in TS case 2 after iteration 80 the solution was not improved, so more than half of the computational time was more or less unnecessary. The solutions of TS and GA are not totally comparable. The sample of scenarios was randomly chosen and differs between the two heuristic. By enlarging the sample size this problem will become less important.

Case 1		Case 2							
Solution	Run time (s)	Solution	Run time (s)	Pop. Size	Elitism	Crossover	Mutation	Hybrids	Gen.
23170	230	47305	828	100	0.40	0.25	0.25	0.10	500
23118	386	47027	1296	100	0.40	0.25	0.25	0.10	750
23118	515	46972	1719	100	0.40	0.25	0.25	0.10	1000
23212	129	47700	862	50	0.40	0.25	0.25	0.10	500
23230	501	47287	1698	200	0.40	0.25	0.25	0.10	500
23287	380	47731	845	100	0.50	0.25	0.25	0.00	500
23283	504	47551	1777	100	0.50	0.25	0.25	0.00	1000
23134	645	47476	1526	100	0.00	0.45	0.45	0.10	500
23134	868	47270	2950	100	0.00	0.45	0.45	0.10	1000

Table 5.12: GA Parameter Analysis

	TS		GA		
	Time (s)	Solution	Time (s)	Solution	Div
Case 1	2624	34378	2876	34735	1.04%
Case 2	14521	68771	6987	69867	1.59%

Table 5.13: Stochastic Results

	# Clusters	# Nodes					
			Mean	Min	Max	Mean excess cap.	Overcap.
Case 1	26	Stage 1	13.19	10	17	3617.2	0.00
		Stage 2	16.23	10	20	2554.3	72.26
Case 2	53	Stage 1	13.34	9	17	4765.5	0.00
		Stage 2	15.75	9	20	5537.9	4622.20

Table 5.14: Stochastic TS Results

5.3.1 Tabu Search

The TS for the stochastic SCP was run with a sample of 50 scenarios and a Tabu List length of 8. Table 5.14, figure 5.5 and 5.6 show the results for case 1 and 2. It can be seen that the allocation of organizations to clusters in the first-stage anticipates on the growth in the second stage. The maximum number of nodes in the first stage is smaller than in the second stage. In case 1 in the second-stage 79 nodes are added and therefore the mean number of nodes increases. In case 2, 128 nodes are added to fulfill the demand, but with respect to case 1 there is more overcapacity. In addition, there is an efficiency loss in terms of mean excess capacity. This is partly due to the fact that nodes can only be added in a cluster in the second-stage and not removed.

If compared with the results in the deterministic part as shown in table 5.8 the number of opened clusters is larger, the mean number of opened nodes is smaller and the mean excess capacity is higher. In the second-stage overcapacity is possible with costs of a penalty. The total amount of penalties in all scenarios is given in the column overcapacity. In case 2 this is much higher than in case 1.

5.3.2 Genetic Algorithm

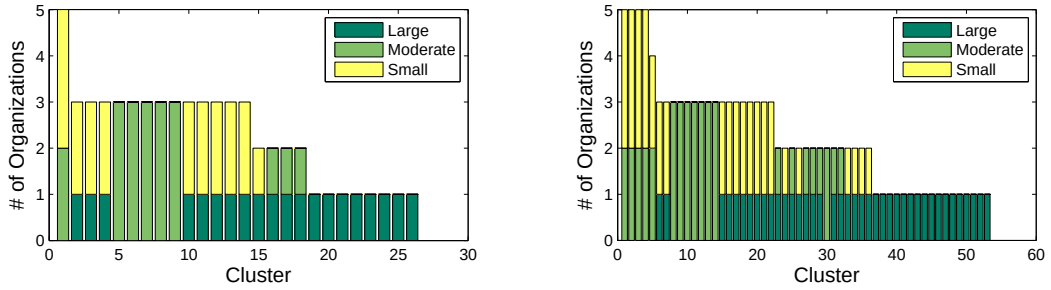
In table 5.15 the results of stochastic GA are shown. The parameters were the same as in the deterministic case. The number of scenarios used was 50. The results show that the number opened clusters was lot a higher then in deterministic GA. Stochastic

	# Clusters	# Nodes					
			Mean	Min	Max	Mean excess cap.	Overcap.
Case 1	28	Stage 1	12.29	10	17	3733.0	0.00
		Stage 2	14.47	10	20	4968.3	20.49
Case 2	55	Stage 1	12.80	10	17	4090.0	0.00
		Stage 2	14.96	10	20	5311.9	1584.60

Table 5.15: Stochastic GA Results

GA anticipates on the growth in the next stage and thus combines less organizations. As a consequences, the number of nodes were lower then before. Also, the mean excess capacity was much higher. Again, this can be explained due to the less combined organizations on a cluster. In case 1 61 additional nodes were added and in case 2 119. GA opened more cluster than TS, since the mean number of nodes is smaller. However, the mean excess capacity is more or less the same. The total overcapacity is relatively low with respect to TS.

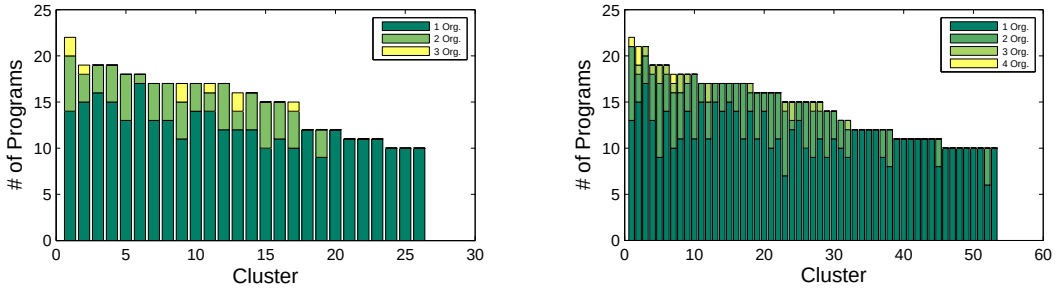
In figure 5.7 the number of organizations in the first-stage on the cluster are presented. This figure confirms that less organizations were clustered. The effect on the overlap is clearly. The overlap on a cluster — as seen in figure 5.8 — was much lower.



(a) Case 1

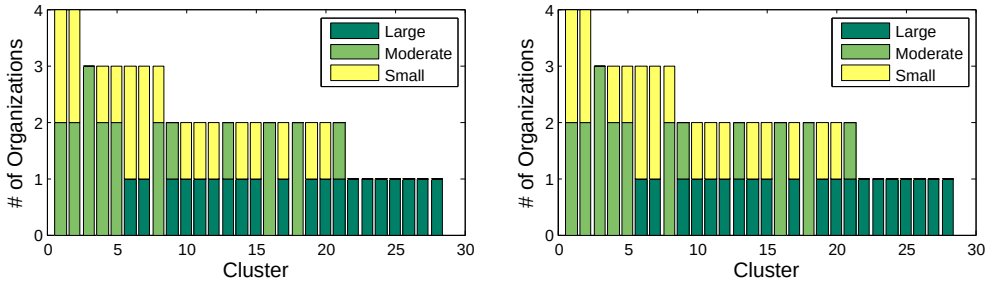
(b) Case 2

Figure 5.5: Stochastic TS results: Number of organizations



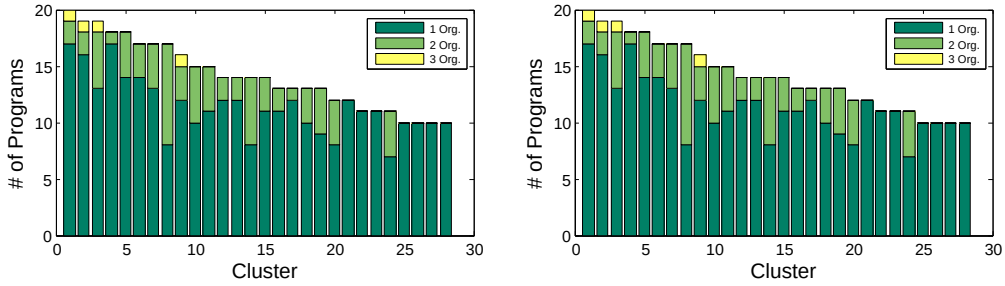
(a) Case 1 (b) Case 2

Figure 5.6: Stochastic TS results: Number of applications



(a) Case 1 (b) Case 2

Figure 5.7: Stochastic GA results: Number of organizations



(a) Case 1 (b) Case 2

Figure 5.8: Stochastic GA results: Number of applications

6 Conclusion

In this thesis the benefits of clustering in SaaS networks were investigated. SaaS networks consist of several clusters and all clients of the SaaS company have to be assigned to a cluster. The clusters differ in installed applications and in size. The size depends on the number of nodes in a cluster. It was investigated how the total costs of a SaaS company could be minimized. Several solution procedures were used and implemented. In the first part of the thesis, the demand of an organization was known-in-advance. However, in the second part, it was assumed to be stochastic.

First, an IP formulation for the *SaaS Clustering Problem* (SCP) was developed. Due to the size of the data sets used in the thesis, the LP/IP solver CPLEX could not found good solution for the test cases in reasonable time. In large data sets, it even struggled to find a feasible solutions. Therefore, several heuristics — known for solving problems like SCP — were developed. These heuristics were; the Lagrange heuristic, Tabu Search and Genetic Algorithm. In the Lagrange heuristic, two Lagrange relaxations were used to simplify the IP formulation. The first relaxation gave better bounds then the LP bound. The second on the other hand, converged to the LP bound. An drawback of these relaxations was the very long computational time. Neither of the two relaxations could calculate good bounds very fast. In the data sets with 120 organizations, a few iteration could take a few hours to calculate. It was assumed that eventually these bounds will correspond with the bounds of the small data sets. In a much faster time, a solution was found with the GA and TS. The advantage of these local search techniques was that it was not required to investigate the whole search space. In all the cases, both local search techniques found a solution in reasonable time. These solution were 15-20% higher then the LP bound and in the first data case only 5% higher then the best Lagrange lower bound. Out of the two, TS systematically outperformed GA by 1-2%.

An extension of the IP formulation was to consider stochastic demand. It was assumed that there were a finite amount of possible scenarios, which gave the opportunity to create a two-stage recourse model. However, due to a large number of scenarios it was nearly possible to solve it with conventional IP solvers like CPLEX, let alone solve the LP bound. As a consequence, the local search techniques — TS and GA — were adjusted

for this extension. In order to reduce the number of possible scenarios in the second stage an idea inspired by SAA was used. A number of randomly drawn scenarios were used to represent the second-stage as good as possible. Again, TS found the best results, while the optimal GA solution was 1-1.5% higher. But now in large data sets there was difference in running time, the computational time was considerably lower in GA.

Some remarks have to be made about this thesis. There were some doubts whether the developed model can be used in the real world. Some assumptions may not be valid and all the relevant costs in the SaaS network may not be incorporated in the model. The IP formulation can possibly be better solved by adding valid equations or changing some restrictions. The computational time of the local search techniques can be improved by more efficient programming or changing to another programming language. The most doubts arise in the stochastic part of this thesis. It was questionable that enough scenarios were used and assumption that the demand was known in advance. In addition, it was not clear that the local search techniques were really the best way to solve this problem.

7 Further Research

Apart from the remarks some other research directions can be given. In the Lagrangian heuristic, the idea was to find good bounds for the SCP. In the thesis, two relaxations were considered but neither gave a sensible solution. However, the first relaxation gave reasonable bounds. In case of the second Lagrange relaxation the remark of convergence to the LP bound was made. However, no formal proof was given why this worst case scenario occurs. Also relevant for both relaxation was the initial α used in the heuristic and the strategy in which α was halved. More research is needed to fully understand these effects. Finally, it maybe a good idea to consider better upper bound for the problem instead of using the solution calculated with the rules of thumb. With these result it maybe possible to obtain better bounds.

As mentioned before, neither of the two local search techniques were efficient implemented, since the main concern of the thesis was getting the heuristics to work instead of fast running time. In further research, the effects of more efficient programming could be studied more intensely. In addition, other heuristics to obtain a solution of the SCP are worth to consider, for example simulated annealing and ant-colony optimization. There is a possibility that these heuristics may perform better than both TS and GA.

The choice of having an application in the created data sets is not correlated with the choice of having another application. It might be interesting to investigate in more detail the choice of applications. Maybe some combinations will never exist in practice. Moreover, it is worth to investigate more data sets of the same type, since the given solutions may not hold in general.

The part were the most further research is needed, is the stochastic extension. The idea inspired by SAA to develop some scenarios for the second stage can be more investigated and especially more scenarios can be used in the stochastic SCP. The influence of the penalty term in the second stage is not clear. By formulating another penalty term, the results may become very different. But most importantly is the need to develop some bounds for the stochastic SCP to check whether the performance of the algorithms is good.

Bibliography

Beasley, J., 1990. Or-library: distributing test problems by electronic mail. Journal of the Operational Research Society 41 (11), 1069 – 1072.

Birge, J., Louveaux, F., 1997. Introduction to Stochastic Programming. Springer, NY, USA.

Bremermann, H. J., 1958. The evolution of intelligence. the nervous system as a model of its environment. Contract No. 477 17, Department of Mathematics, University of Washington, Seattle.

Chaudhry, S., He, S., Chaudhry, P., May 2003. Solving a class of facility location problems using genetic algorithms. Expert Systems 20 (2), 86 – 91.

Correa, E., Steiner, M., Freitas, A., Carnieri, C., 2004. A genetic algorithm for solving a capacitated p-median problem. Numerical Algorithms 35, 373 – 388.

de Jong, K., 1975. An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation.

de Jong, W., 2009. The impact of a multi-cluster environment on operational costs. Master's thesis, Erasmus University Rotterdam.

Fisher, M., 2004. Comments on "the lagrangian relaxation method for solving integer programming problems". Management Science 50, 1872 – 1874.

Fogel, L., Owens, A., Walsh, M., 1966. Artificial Intelligence through Simulated Evolution. John Wiley, New York USA.

Gendreau, M., 2003. Handbook of Metaheuristics. Kluwer Academic Publishers, Dordrecht, Ch. 2, pp. 51 – 67.

Geoffrion, A., Bride, R. M., 1978. Lagrangean relaxation applied to capacitated facility location problems. IIE Transactions 10 (1), 40 – 47.

Ghiani, G., Guerriero, F., Musmanno, R., 2002. The capacitated plant location problem with multiple facilities in the same site. Computers & Operations Research 29, 1903 – 1912.

Ghiani, G., Laporte, G., Musmanno, R., 2004. Introduction to Logistics Systems Planning and Control. John Wiley & Sons Ltd, West Sussex, England.

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

Hajiaghayi, M., Mahdian, M., Mirrokni, V., 2002. The facility location problem with general cost functions. Networks 42, 42 – 47.

Holland, J., 1975. Adaptation in Natural and Aritificial Systems. The MIT Press, Cambridge, MA, USA.

Kivits, N., 2009. 'saas-opbrengst groeit in 2009 met 22 procent'. On-Line, accessed: May 12, 2009.
URL `http://www.computable.nl/artikel/ict_topics/saas/2945236/2333364/saasopbrengst-groeit-in-2009-met--procent.html`

Kratica, J., Tosic, D., Filipovic, V., Ljubic, I., 2001. Solving the simple plant location problem by genetic algorithm. RAIRO Operations Research 35, 127 – 142, communicated P. Tolla.

Michel, L., Hentenryck, P. V., 2004. A simple tabu search for warehouse location. European Journal of Operational Research 157, 576 – 591.

Mitchell, M., 1998. An introduction to genetic algorithms, reprint, illustrated Edition. MIT Press, USA.

Nitu, 2009. Configurability in saas (software as a service) applications. In: ISEC '09: Proceeding of the 2nd annual conference on India software engineering conference. ACM, New York, NY, USA, pp. 19 – 26.

Ravi, R., Sinha, A., August 2006. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. Mathematical Programming 108 (1), 97 – 114.

Rechenberg, I., 1993 1973. Evolutions strategic: Optimierung technischer Systeme nach Prinzipen der biologischen Evolution, 2nd Edition. Frommmann-Holzboog Verlag, Stuttgart.

Reeves, C., 2003. Handbook of Metaheuristics. Kluwer Academic Publishers, Dordrecht, Ch. 3, pp. 51 – 67.

Rolland, E., Schilling, D., Current, J., 1996. An efficient tabu search procedure for the p-median problem. European Journal of Operational Research 96, 329 – 342.

Schwefel, H., 1977. Numerische Optimierung von Computer-modellen mittels der Evolutionsstrategie. Birkhäuser Verlag, Basel.

Shapiro, A., Philpott, A., March 2007. A tutorial on stochastic programming. Online.

Sun, M., May 2008. A tabu search heuristic procedure for the capacitated facility location problem, university of Texas at San Antonio, College of Business Working Paper Series.

Waters, B., 2005. Software as a service: A look at the customer benefits. Journal of Digital Asset Management 1, 32 – 39.

Wolsey, L., 1998. Integer Programming. John Wiley & Sons, USA.

Wu, L., Zhang, X., Zhang, J., 2006. Capacitated facility location problem with general setup cost. Computers & Operations Research 33, 1226 – 1241.

Zhang, H., Sun, G., March 2002. Feature selection using tabu search method. Pattern Recognition 35 (3), 701 – 711.