A Local Search Algorithm for Clustering in Software as a Service networks

Jelmer P. van der Gaast, Cornelius A. Rietveld, Adriana F. Gabor, and Yingqian Zhang

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2011-004-LIS
Publication	March 2011
Number of pages	24
Persistent paper URL	http://hdl.handle.net/1765/22723
Email address corresponding author	nrietveld@ese.eur.nl
Address	Erasmus Research Institute of Management (ERIM) RSM Erasmus University / Erasmus School of Economics Erasmus Universiteit Rotterdam P.O.Box 1738 3000 DR Rotterdam, The Netherlands Phone: + 31 10 408 1182 Fax: + 31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

Bibliographic data and classifications of all the ERIM reports are also available on the ERIM website: www.erim.eur.nl

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

ABSTRACT AND KEYWORDS	
Abstract	In this paper we present and analyze a model for clustering in networks that offer Software as a Service (SaaS). In this problem, organizations requesting a set of applications have to be assigned to clusters such that the costs of opening clusters and installing the necessary applications in clusters are minimized. We prove that this problem is NP-hard, and model it as an Integer Program with symmetry breaking constraints. We then propose a Tabu search heuristic for situations where good solutions are desired in a short computation time. Extensive computational experiments are conducted for evaluating the quality of the solutions obtained by the IP model and the Tabu Search heuristic. Experimental results indicate that the proposed Tabu Search is promising.
Free Keywords	Tabu Search, integer programming, software as a service, complexity theory
Availability	
Classifications	

A Local Search Algorithm for Clustering in Software as a Service networks

Jelmer P. van der Gaast[a,*], Cornelius A. Rietveld[b], Adriana F. Gabor[b], Yingqian Zhang[b]

[a] *Rotterdam School of Management, Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*
[b] *Erasmus School of Economics, Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*

Abstract

In this paper we present and analyze a model for clustering in networks that offer Software as a Service (SaaS). In this problem, organizations requesting a set of applications have to be assigned to clusters such that the costs of opening clusters and installing the necessary applications in clusters are minimized. We prove that this problem is NP-hard, and model it as an Integer Program with symmetry breaking constraints. We then propose a Tabu search heuristic for situations where good solutions are desired in a short computation time. Extensive computational experiments are conducted for evaluating the quality of the solutions obtained by the IP model and the Tabu Search heuristic. Experimental results indicate that the proposed Tabu Search is promising.

Keywords: Tabu Search, Integer programming, Software as a Service, Complexity theory

1. Introduction

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. Companies that offer software manage and maintain their software centrally and offer the software to customers via Internet. In this way, the costs for purchasing the software and maintaining it are shared among several customers. It was only since 2005-2006 that the SaaS wave gained momentum. The reason for this is twofold: on one hand, the Internet became high-speed and affordable, and on the other hand, customers became comfortable with doing business on the web. They realized that good backup and fault tolerant practices made data more secure and reliable with the SaaS offering company rather than with their own enterprise. In addition, due to the faster upgrade cycle of SaaS software and small deployment time, there is no need for customers to spend time and money in upgrades and deployment of the software. As for the recent days, the market for SaaS software is rapidly growing. The revenue of SaaS was forecast to reach $9.2 billion in 2010, up 15.7 percent from 2009 revenue of $7.9

*Corresponding author. Tel.: +31 10 408 2591; fax: +31 10 408 9014
Email address: jgaast@rsm.nl (Jelmer P. van der Gaast)

billion (Mertz et al., 2010). Till 2014 the market is expected to grow with a 15.8% compound annual growth rate.

Researchers also show their increasing interests on SaaS from many different aspects, ranging from its adoption strategy and analysis (Xin and Levina, 2008), software development (Bennett et al., 2000), to the general business model (Sääksjärvi et al., 2005). However, relevant optimization problems have been rarely mentioned and studied by the existing literature. In this paper, we look into an important SaaS network optimization problem, i.e., the way SaaS networks should be designed in order to satisfy the demand of the customers, while minimizing the cost for the SaaS offering company.

We focus on clustering in SaaS networks. Clustering means deciding the number and capacity of clusters needed in order to assign each organization to only one cluster. The overall objective is that the costs of opening clusters and installing the requested applications are minimized. Clustering reduces the costs of hardware and software installation for the SaaS offering company, since organizations that share a cluster and request the same application share the costs of installing and maintaining this application.

The organization of this paper is as follows. In Section 2 the problem is formulated, related work is discussed and the relationship of the *SaaS Clustering Problem* (SaaSCP) with other combinatorial problems is summarized. In Section 3 the computational complexity of the problem is studied and an integer linear program is proposed. Section 4 contains the description of a Tabu Search heuristic, the results of which are extensively analyzed in Section 5 via computational experiments. The final section contains concluding remarks and possible further research topics.

2. Problem and Related Work

The SaaS clustering problem (SaaSCP) can be described as follows. Let I be a set of m organizations, J a set of n clusters, and K a set of applications. For each organization $i \in I$, one is given a set $K_i \subseteq K$ of applications needed and the demand d_{ik} for each application $k \in K_i$. The demand of an organization for an application is defined as the number of users in the organization times the number of transactions per second for that application.

The clusters have different capacities q_l, $l \in L$. We assume that the total demand of a single organization does not exceed the maximal capacity of a cluster. The cost of opening a cluster of capacity q_l is e_l. The cost of installing an application k on a cluster is the same for every cluster and is equal to c_k.

An allocation $a : I \mapsto J$ of organizations to clusters will be called *feasible* if: (1) for every organization $i \in I$, all its required applications are installed in cluster $a(i)$; and (2) organizations are assigned only to open clusters and the demand of all the organizations installed on a cluster does not exceed the capacity of the cluster. Otherwise an allocation is called *infeasible*.

The goal in the SaaSCP is to decide the number of clusters that have to be opened and their capacity, together with a feasible allocation of organizations to clusters such that the total costs, i.e., the costs of

opening clusters and installing applications, are minimized.

2.1. Related Work

The SaaSCP is closely related to several classical NP-hard problems in the fields of Operations Research and Computer Science. From the field of OR, the problem is strongly related to the multiproduct capacitated facility location problem (MPCFL). In this problem, given a set of sites where facilities may be opened and a set of clients with different desired products, one has to decide the location and the type of the open facilities, together with the assignments of clients to facilities, such that the total costs are minimized. If all the clients demand the same product and all the facilities are of the same type, the problem reduces to the classical capacitated facility location problem (CFLP). The SaaSCP can be viewed as a single source MPCFL where the transportation costs are zero. The applications needed by an organization in the SaaSCP correspond to the products in the MCFLP and the facilities to the clusters. The requirement that an organization should be assigned to only one cluster is similar to the single source requirement in the facility location literature.

The literature on CFLP is vast and many heuristics and exact algorithms have been developed for finding good solutions (Sridharan, 1995). For the single source case, the majority of the proposed solution methods are based on Lagrangian relaxations (Pirkul, 1987; Beasley, 1993; Sridharan, 1993; Rönnqvist et al., 1999). Local search techniques enhanced with specialized procedures to tackle the capacity constraints have also been reported to be successful for this type of problems. In Delmaire et al. (1999) results on hybrid algorithms combining Tabu search with GRASP are reported for instances with up to 30 clients and 90 location sites for facilities. In Ahuja et al. (2004) it is shown that a multi-exchange heuristic works well on instances with up to 30 clients and 200 facilities. A heuristic based on randomized rounding, that can solve instances of up to 1000 clients and 1000 facilities is described in Barahona and Chudak (2005). Exact solutions based on branch and bound are proposed in Neebe and Rao (1980) and Holmberg et al. (1999).

The multiproduct (multi-commodity) variant of the CFLP has received less attention in the literature. In Mazzola and Neeb (1999) a Lagrangian heuristic is proposed for solving the multi sourcing variant of MCFLP in which several types of facilities may be opened at a location and in Lee (1991) the problem is solved via Bender's decomposition. More complex variants of MCFLP are studied in Pirkul and Jayarama (1998), where the problem is extended to two echelons and a solution is found by a Lagrangian heuristic and in Melo et al. (2005), where the problem is embedded in a model of a supply chain network and solved via a commercial IP solver.

For the single source variant of MCFLP problem to which the SaaSCP is most related there was no literature found. The reason may be that in a logistic network it is not common that a client requires all its products to be delivered from only one facility.

Another strongly related problem to SaaSCP is the generalized bin packing problem (GBPP). In this problem, a set of objects of different volumes have to be assigned to bins of different capacities. Costs are

incurred if a bin is being used. It is easy to see that the GBPP is a special case of the SaaSCP, in which all organizations need the same applications and the capacity of a cluster is given. For the GBPP, greedy algorithms are proposed and their worst case behavior is analyzed in Kang and Park (2003). In Crainic et al. (2011), the authors present extensive numerical studies of several heuristics inspired from known heuristics for the classical bin packing and knapsack problems.

Through the set of applications shared by several organizations, the SaaSCP is related to another classical problem in computer science, namely the graph partitioning problem (GP). GP consists of dividing a graph into disjoint parts such that all parts satisfy some size constraints and the size of the edge cuts shared between parts is minimized. The problem is known to be NP-complete (Garey and Johnson, 1979). The SaaSCP is similar to GP, in the sense that in the SaaSCP, we want to reduce the cost by not assigning the organizations with many shared applications to different clusters. We will use GP to prove the hardness of SaaSCP in the next section. Popular heuristics for solving the graph partitioning problem include for instance the Kernighan–Lin algorithm (Kernighan and Lin, 1970).

As we have seen, the SaaSCP combines features characteristic of several NP-hard problems. In this paper, we propose to find a solution of SaaSCP by a Tabu Search heuristic enhanced with a randomized procedure to reduce the size of the neighborhood. Tabu Search has been previously used successfully for other facility location problems such as the uncapacitated facility location problem (Ghosh, 2003; Michel and Hentenryck, 2004), the p-median problem (Rolland et al., 1996; Ghosh, 2003; Sun, 2006) and the capacitated facility location problem (Delmaire et al., 1999). In the same time, Lodi et al. (1999) and Crainic et al. (2009) incorporate Tabu Search in algorithms designed to solve two and three dimensional bin packing problem with promising results.

In next section we formalize the SaaS clustering problem and analyze the complexity of the problem.

3. Computational Complexity and Integer Programming Formulation

3.1. Computational Complexity

We may consider the generalized bin packing problem as a special case of the SaaSCP by assuming that all organizations need the same set of applications, and then conclude the SaaSCP is also NP-complete. However, this does not give sufficient insights into the computational difficulty of the SaaSCP, since in practice it rarely occurs that all organizations require the exactly same applications. Hence, we derive the hardness of the SaaSCP from another NP-complete problem, the graph partitioning problem, and show that the overlapping applications between organizations play an important role in the difficulty of solving the SaaSCP. The graph partitioning (GP) problem is described as follows:

Graph partitioning (Garey and Johnson, 1979). Given a graph $G = (V, E)$, weights $w(v) \in Z^+$ for each $v \in V$ and $l(e) \in Z^+$ for each $e \in E$, positive integers O and R, is there a partition of V into disjoint sets

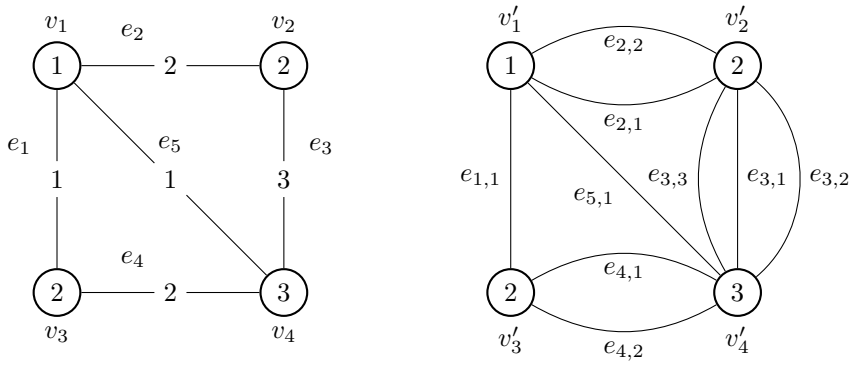


Figure 1: GP is reduced to SaaSCP. The optimal solution for the GP instance (with $O = 6$) is the partition $V_1 = \{v_3\}$, $V_2 = \{v_1, v_2, v_4\}$, with the minimal edge cuts 3. The figure on the right hand side is the constructed SaaSCP instance, where given the cluster's maximal capacity of 6, the optimal allocation is v'_3 to one cluster, and $\{v'_1, v'_2, v'_4\}$ to another cluster. This gives a minimal cost of 12.

$V_1, V_2, \ldots, V_n$ such that $\sum_{v \in V_i} w(v) \leq O$ for $1 \leq i \leq n$ and such that if $\hat{E} \subseteq E$ is the set of edges that have their two endpoints in two different sets, $\sum_{e \in \hat{E}} l(e) \leq R$?

We now show that the SaaSCP problem is also NP-complete.

Theorem 1. *Given the SaaSCP, the problem of deciding the existence of an assignment of m organizations to n clusters of a total cost at most X is NP-complete.*

Proof. First we show the problem is in NP. Given a problem instance of the SaaSCP and an integer X, we can verify in polynomial time whether an allocation of organizations to clusters is feasible, and whether the total cost is smaller than X. Next we prove that the SaaSCP is NP-hard by reducing from GP.

Given a GP problem with graph $G = (V, E)$ and integers O and R, we construct a SaaSCP network $G' = (V', E')$, which has a feasible allocation of organizations V' to clusters with cost X if and only if there is a feasible partition in G such that $\sum_{e \in \hat{E}} l(e) = R$. The construction is described as follows (see Figure 1 for an example). For each vertex $v \in V$ in G, we create a vertex $v' \in V'$ in G' representing an organization. For each edge $e_k \in E$ between two vertices v_i and v_j ($v_i, v_j \in V$) with weight $l(e_k)$, in G' we create $l(e_k)$ number of edges $e_{k,1}, e_{k,2}, \ldots, e_{k,l(e_k)}$ between two organizations $v'_i \in V'$ and $v'_j \in V'$. Each such edge e' in E' represents a unique application with installation cost $l(e') = 1$. The set of required applications $K_{v'_i}$ of each organization v'_i is a set of edges that connect to v'_i, i.e., $K_{v'_i} = \cup_{e'=(v'_i, v'_j), v'_j \in V'} e'$. In this way, the number of edges between v'_i and v'_j in G' specifies the number of overlapping applications between two organizations v'_i and v'_j. And the total number of unique applications in the SaaSCP is the number of edges in G', i.e., $|E'|$. Further, for each $v' \in V'$, define $w(v') = w(v)$ ($v \in V$) denoting the total demand $D_{v'}$ of organization v' for its applications. In addition, let the maximal capacity of each cluster be O and the cost of opening clusters be 0. And finally, define $X = |E'| + R$.

Suppose there is a feasible partition $V_1, \ldots, V_n$ for a given GP problem with $\sum_{v \in V_i} w(v) \leq O$ and

$\sum_{e\in\hat{E}} l(e) \leq R$, where $\hat{E}$ is the set of cut edges that have their two endpoints in two sets. Then in the constructed SaaSCP problem, the vertices V' are partitioned into n disjoint sets $V'_1, \ldots, V'_n$, representing in the allocation a, the organizations are assigned to n clusters. Thus in a, each cluster j has a set of organizations $v' \in V'_j$, and the applications installed on j are a set of edges that are connected to organizations $v' \in V'_j$. Because the partitioning of GP is feasible, we have $\sum_{v\in V_j} w(v) \leq O$. It implies $\sum_{v'\in V'_j} D_{v'} = \sum_{v\in V_j} w(v) \leq O$, that is, the total demand of organizations in every cluster V'_j is less than its maximal capacity O. The number of applications that need to be installed twice (i.e., in two different clusters) is represented by the size of the cut edges, which is equal to $\sum_{e\in\hat{E}} l(e)$. All other applications are required by organizations belonging to the same cluster, and therefore, they need to be installed only once. Since there is no cost of opening clusters, the cost of allocation $C(a)$ is simply the total number of applications that need to be installed, i.e., $C(a) = |E'| + \sum_{e\in\hat{E}} l(e) \leq |E'| + R = X$.

Similarly, it is not difficult to check that if there is a feasible allocation a which assigns the organizations to n disjoint clusters, where the demand of organizations in each cluster is smaller than its capacity O, with the total of cost $C(a) \leq X$. In the corresponding GP problem, there exists a partitioning of V to n sets, and the total weight of the cut edges is $\sum_{e\in\hat{E}} l(e) = C(a) - |E'| \leq X - |E'| = R$.

Therefore, SaaSCP is NP-complete. □

Since the problem is NP-complete, there exists no polynomial algorithm for finding an optimal solution, unless $P = NP$. In next subsection, we give an Integer Programming model for the SaaSCP, which will be used in our computational experiments to output optimal solutions for small problem instances and to assess the performance of CPLEX and of the Tabu Search heuristic on this problem.

3.2. On an Integer Program Formulation

In this section we first propose a straightforward Integer Program (IP) to model the SaaSCP. This program may, however, lead to a high running time due to the symmetry of the optimal solutions. Subsequently, we will discuss how this symmetry can be avoided by adding a new set of constraints to the initial program.

The IP model of SaaSCP makes use of the binary decision variables (x, v, z) which are defined as follows:

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

$$v_{jl} = \begin{cases} 1 & \text{if cluster } j \text{ is opened and capacity } l\text{has been chosen for it;} \\ 0 & \text{otherwise.} \end{cases}$$

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

The SaaSCP problem can be now formulated as an IP as follows:

$$\min \quad \sum_{j\in J}\sum_{l\in L} e_l v_{jl} + \sum_{k\in K}\sum_{j\in J} c_k z_{kj} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ij} = 1 \qquad i \in I \tag{2}$$

$$\sum_{i\in I}\sum_{k\in K_i} d_{ik} x_{ij} \leq \sum_{l\in L} q_l v_{jl} \qquad j \in J \tag{3}$$

$$x_{ij} \leq z_{kj} \qquad k \in K,\, i \in I,\, j \in J \tag{4}$$

$$\sum_{l\in L} v_{jl} \leq 1 \qquad j \in J \tag{5}$$

$$x_{ij}, z_{kj}, v_{jl} \in \{0,1\} \qquad i \in I,\, j \in J,\, k \in K,\, l \in L \tag{6}$$

The objective function (1) represents the total costs, which consist of the costs of opening clusters of a certain capacity and the costs of installing applications. Constraints (2) ensure that each organization is assigned to exactly one cluster. Constraints (3) state that a cluster cannot serve more than the installed capacity. They also imply that one cluster should be opened if at least an organization is assigned to it. Constraints (4) ensure that if an organization $i \in I$ is assigned to a cluster $j \in J$, all its applications are also installed on that cluster. Finally, constraints (5) are added to the program to ensure that each cluster gets assigned only one value of the capacity.

Note that since the objective is to minimize the costs and $x_{ij} \in \{0,1\}$, the constraints $z_{kj} \in \{0,1\}$ can be relaxed to $z_{kj} \geq 0$. However, even in this case, solving the above IP program directly with CPLEX proved to be quite difficult for large instances. One of the causes we have identified is the symmetry of the set of feasible solutions. Since the costs of opening clusters with same capacity are equal, every feasible solution leads to another $n!$ other feasible solutions, in which only the numbering of the clusters changes, but not the set of organizations assigned to the same cluster. In order to reduce computation time due to this symmetry, one may add symmetry breaking constraints, see for example Sherali and Smith (2001) and Jans (2009).

In this paper we have opted for constraints that ensure that organization i can be assigned to cluster j only if organizations $1, \ldots, i-1$ are assigned to clusters $1, \ldots, j-1, j$, in other words, that the vectors $(x_{1j}, \ldots, x_{mj})$ are lexicographically greater ($\geq_L$) than $(x_{1,j+1}, ..., x_{m,j+1})$ for any cluster $j \geq 1$. To impose that organization number 1 is assigned to cluster number 1, we add the following constraints:

$$x_{11} \geq x_{12} \geq \cdots \geq x_{1n}$$

To ensure that organization number 2 gets assigned to either cluster 1 or 2, we add the constraints:

$$2x_{11} + x_{21} \geq 2x_{12} + x_{22} \geq \cdots \geq 2x_{1n} + x_{2n}$$

Observe that the first inequality ensures that organization 2 can be assigned to cluster 2 only if organization 1 has been assigned to cluster 1, whereas the other inequalities ensure that organization 2 should be

assigned to cluster 1 or 2 and not to clusters $3, 4, ..., |J|$.

In general, in order to ensure that organization i can be assigned to cluster j only if organizations $1, \ldots, i-1$ are assigned to clusters $1, \ldots, j-1, j$, we add the following constraint:

$$\sum_{k=1}^{i} 2^{i-k} x_{k,j-1} \geq \sum_{k=1}^{i} 2^{i-k} x_{kj} \qquad \forall i \in I, \quad \forall j \in J \backslash \{1\} \tag{7}$$

This constraint is equivalent to $(x_{1j}, ..., x_{mj}) \geq_L (x_{1,j+1}, ..., x_{m,j+1})$, as it is proved in proposition 2 in Sherali and Smith (2001). Note that the result of this Proposition is independent of the interpretation given to these vectors, hence it can be applied in the case of SaaSCP.

We will test the performance of this IP program in CPLEX in Section 5. As we will see, for large problems, the running time of this program is relatively high. As an alternative, in the next section we propose a Tabu Search heuristic that can be used when feasible solutions close to optimal are desired in a short running time.

4. Tabu Search Heuristic

A Tabu Search algorithm (TS) is a local search algorithm that allows non improving moves when a local optimum is encountered, in the hope that in this way the solution will be improved. Cycling back to previously visited solutions is prevented by a list of forbidden moves, called the Tabu list (Gendreau, 2003). A comprehensive introduction to TS can be found in Glover and Laguna (1997). We now describe a Tabu Search heuristic for the SaaSCP.

4.1. Terminology

Before describing the Tabu Search algorithm in detail, we briefly introduce some terminology.

- A *move* from allocation a to allocation a' corresponds to the set of subsequent assignments of organization to clusters, that lead from a to a'. In this paper we will consider the following two types of moves: *shift* and *interchange* moves. In a shift move, denoted by $\varphi_s(a, a')$, an organization is reassigned to an already opened cluster or to a new cluster. If a new cluster is opened, a cost equal to e_l is incurred, depending on the chosen capacity. When an organization i is placed on a cluster that does not contain all the applications in K_i, installation costs for the extra needed applications are incurred. If after the shift operation there is a cluster to which no organization is assigned to, the cluster is closed. Also, if there is an application which is not requested anymore by an organization, the installation costs for this cluster are correspondingly reduced. Observe that a move can be performed even if the allocation obtained does not satisfy the capacity constraints and is therefore infeasible. In an interchange move, denoted by $\varphi_{int}(a, a')$, two organizations switch the clusters they are assigned to. If by doing this, an organization is assigned to a cluster on which not all its applications are present, new applications will

be installed . Applications previously installed on a cluster and not required anymore after the switch are removed from the cluster. In both cases, the installation costs are modified accordingly.

- The *neighborhood* of allocation a, denoted by $\mathcal{N}(a)$, is the set of allocations that can be constructed from a by a shift or an interchange move. Note that since a move does not necessarily lead to a feasible allocation, a neighborhood may contain infeasible allocations.

- *Tabu lists*: In our algorithm we maintain two Tabu lists, one of fixed size and one of variable size. The Tabu lists are lists of forbidden moves, with the primary goal of preventing the algorithm from cycling. Both Tabu lists are implemented as first in first out (FIFO) queues.

- *Aspiration Criterion (Level)*: The aspiration criterion is a condition that helps in deciding whether a move is Tabu or not. If a move leads to an allocation that satisfies the aspiration criterion, it cannot be declared Tabu. The simplest and most commonly used aspiration criterion consists in allowing a move if it results in a solution with a better objective value than the current best-known solution.

- *Stopping Criterion*: The Tabu search will stop after a predefined number of iterations T.

In the next subsection we present a Tabu search algorithm, which will subsequently be fastened by a neighborhood reduction technique.

4.2. Basic Algorithm

The algorithm, which is described in Algorithm 1 proceeds through several phases that try to find a feasible solution with total costs close to the optimal value.

1. Construction phase. In this phase, an initial feasible allocation a^{ini} is constructed. We assume that the number of available clusters is sufficient, that is, it is at least equal to the number of organizations. In this case an initial feasible solution can be easily obtained by opening a dedicated cluster for each organization. Note that this assumption is realistic in the context of SaaS, because in practice for a software offering company, the benefit of opening an extra cluster is usually higher than the loss incurred by refusing an organization as client. However, for the purpose of completeness, we will also discuss how to deal with the case in which the number of available clusters is strictly smaller than the number of organizations at the end of this subsection, in Remark 1.

We initialize the current solution a^{cur} and the so far best solution a^{best} as a^{ini}. Both Tabu lists TL_1 and TL_2 are at this moment empty, which means that all the moves are allowed.

2. Improvement phase. In this phase the algorithm searches in the neighborhood $\mathcal{N}(a^{cur})$ of the current best solution a^{cur} for a new candidate for the best solution. This candidate is either a feasible solution or an infeasible solution which violates some of the capacity restrictions. The search is conducted in terms of the value associated to an allocation. This can be explained by the following three steps.

2a. Calculating the value of allocations in the neighborhood $\mathcal{N}(a^{cur})$. The value $v(a)$ of an allocation a is defined as

$$v(a) = f(a) + p_1(a) + p_2(a) \tag{8}$$

where $f(a)$ represents the costs associated to a, $p_1(a)$ a penalty function for exceeding the maximal capacity of the cluster, and $p_2(a)$ a penalty function used to diversify the search. Next we describe all the components of the value function in more detail.

The costs $f(a)$ for a given allocation a comprise the costs for opening the clusters in $a(I)$ and the costs of installing all the necessary applications. Clearly, for a cluster j, the capacity required is equal to $R(j) = \text{argmin}\left\{l \mid \sum_{i:a(i)=j} \sum_{k \in K_i} d_{ik} \leq q_l\right\}$. Hence,

$$f(a) = \sum_{j \in a(I)} \sum_{k \in \cup_{i:a(i)=j} K_i} c_k + e_{R(j)} \tag{9}$$

The penalty p_1 for exceeding the maximal capacity of the cluster, q_{max}, is defined as

$$p_1(a) = \beta \sum_{j \in J} \left[\sum_{i \in I:a(i)=j} \sum_{k \in K_i} d_{ik} - q_{max} \right]^+ \tag{10}$$

where $[b]^+ = \max(0, b)$.

The parameter β is dynamically adjusted during the search in order to explore more intensively solutions that do not violate the constraints too much. Before calculating $p_1(a)$, parameter β is divided by $1+\theta$ if the current solution a^{cur} is feasible, and multiplied by $1+\theta$ if a^{cur} is infeasible. In this way, if β is low, the algorithm can visit infeasible solution, while the search is directed towards exploring feasible solutions if β is high.

The penalty function $p_2(a)$ is used to diversify the search, in order to drive the algorithm towards unexplored regions. One way of doing this is by favoring infrequently used moves. Following Cordeau and Laporte (2003), we define $p_2(a)$ as

$$p_2(a) = \lambda f(a) \sqrt{|I| \cdot |J|} \rho_{\phi(a^{cur},a)}/t \tag{11}$$

where $f(a)\sqrt{|I| \cdot |J|}$ is the scaling factor and t is the number of the current iteration. Parameter λ controls the intensity of the diversification and $\rho_{\phi(a^{cur},a)}$, where $\phi(a^{cur}, a)$ is a shift or interchange move, is a parameter that penalizes frequently made moves. We define $\rho_{\phi(a^{cur},a)}$ as follows. For $i \in I$ and $j \in J$, let n_{ij} be the number of times organization i is placed on cluster j during the search. For shift moves $\phi_s(a^{cur}, a)$ by which organization i is moved to cluster j, $\rho_{\phi_s(a^{cur},a)} = n_{ij}$. For interchange moves that switch the clusters of organizations i to cluster i', $\rho_{\phi_{int}(a^{cur},a)} = n_{ia(i)} + n_{i'a(i')}$. Observe that $p_2(a)$ penalizes moves that were frequently made and result in high value.

2b. Choosing the best solution in the neighborhood. In this phase, the value of the current allocation a^{cur} is compared with the values of the other allocations in $\mathcal{N}(a^{cur})$ and the allocation with the lowest value among the allocations obtained by a move that it is not in a Tabu list or which is Tabu but satisfy the aspiration criterion is chosen. Denote by $\tilde{a}$ the solution found by the neighborhood search.

2c. Modification of the Tabu list. In this phase new moves are added to the Tabu list and old moves are deleted. The first Tabu list TL_1 has a maximal fixed size α, whereas the second Tabu list TL_2 has a variable size.

For a shift move $\phi_s(a^{cur}, \tilde{a})$ that assigns organization i to a cluster $j \neq a^{cur}(i)$, the move by which i is assigned to $a^{cur}(i)$ becomes Tabu. This move is added to TL_1. For an interchange move $\phi_{int}(a^{cur}, \tilde{a})$ by which organizations i and i' interchange clusters, both the moves that assign i to $a^{cur}(i)$ and i' to $a^{cur}(i')$ become Tabu and are added to TL_1. If the size of TL_1 is larger than α, moves are deleted in a FIFO manner.

The second Tabu list TL_2 depends on the list TL_1 and is updated at every change of TL_1. Suppose that organization i was moved from cluster j to j' and the move that assigns i to j became Tabu. Then, all the moves that assign an organization i' on j to cluster j' are declared Tabu and added to TL_2. The role of these moves is to prevent recreating the same clusters during the search.

3. Intensification and updating the current solution. If a^{best} remained unchanged in the last τ iterations (no allocation of lower cost was found), the search is intensified by choosing as a^{cur} an allocation that is more likely to lead the search in a promising region. The new allocation a^{cur} is obtained from a^{best} by performing several subsequent shift moves which aim to reduce the amount of clusters by one. This is done as follows. Start with allocation a^{best}. Suppose that cluster j and j' are the clusters with the lowest and second lowest demand, respectively. If possible, shift the organization with the lowest demand on j to cluster j'. If the capacity restriction of j' is not violated, decide again the clusters with the smallest demand and repeat the shift. Stop the intensification when the number of cluster has been reduced by one or a shift move leads to an infeasible allocation.

In this step also the allocations a^{cur} and a^{best} are updated. $\tilde{a}$ becomes the new $a^{cur}$and also the new a^{best}, provided that $f(\tilde{a}) < f(a^{best})$ and $\tilde{a}$ is feasible.

4. Check the stopping criteria. Next, increase the iteration counter $t = t + 1$ and if the maximum number T of iterations has not been reached, proceed with Step 1. Otherwise the algorithm is terminated with a^{best} as the allocation with the lowest cost for the SaaSCP.

Remark 1 If the number of available clusters is strictly smaller than the number of organizations, one can introduce dummy clusters at very high costs and run the above algorithm for the new instance of the problem. If in the best solution some organizations are assigned to the dummy clusters, the algorithm was not able to find a feasible solution for the problem without the dummy clusters. Results on using this

approach are presented in Section 5.3.6. Note, however, that in practice the situations in which the number of available clusters is restricted are rare.

Algorithm 1: Tabu Search algorithm

```
   // 1. Construction Phase
1  Generate a^ini;
   // 2. Improvement Phase
2  Let a^cur = a^best = a^ini, TL_1 = TL_2 = ∅, t = 0;
3  repeat
4  |  Construct the neighborhood N(a^cur) of allocation a^cur;
5  |  if (Neighborhood Reduction rule active) then
6  |  |  Reduce N(a^cur) by choosing a subset of allocations from the neighborhood;
7  |  end
   |  // 2a. Calculating the value of allocations in the neighborhood
8  |  Calculate for every a in N(a^cur) its value, v(a) = f(a) + p_1(a) + p_2(a);
   |  // 2b. Choose the best solution in the neighborhood
9  |  Choose the allocation ã ∈ N(a^cur) with the least v(a) for which φ(a^cur, ã) ∉ TL_1 or TL_2 or for
   |  which the aspiration criterion f(ã) < f(a^best) is satisfied;
   |  // 2c. Modify the Tabu list
10 |  Update TL_1 = TL_1 ∪ {φ(a^cur, ã)} and if necessary, remove the first elements of TL_1 till |TL_1| = α;
11 |  Update TL_2 using the latest move φ(a^cur, ã);
   |  // 3. Intensification and update the current solution
12 |  if (Intensification needed) then
13 |  |  Intensify the search using a^best;
14 |  end
15 |  Let a^cur = ã, and let a^best = ã if ã has lower cost than a^best and is feasible;
16 |  Adjust infeasibility parameter β with factor θ;
17 |  t = t + 1;
   |  // 4. Check the stopping criteria
18 until t > T;
19 Output a^best;
```

4.3. Neighborhood Reduction

In this subsection, we analyze the running time of the proposed TS algorithm and present a method to reduce it. The effect of the proposed method on the quality of the solution will be discussed in Subsection 5.3.2.

Note that a large part of the running time stems from evaluating the value of all the elements in the neighborhood of a^{cur}. Recall that the neighborhood of an allocation is obtained by shift and interchange moves, and therefore contains $|I|(|J|-1)+\binom{|I|}{2} = O(m \cdot \max(m,n))$ allocations that need to be evaluated in an iterations. A reduction in the neighborhood size that would not restrict the search from exploring promising regions would thus have a big influence on the running time.

The neighborhood reduction technique we use in this paper is described as follows. Instead of evaluating all the allocations in $\mathcal{N}(a^{cur})$, the algorithm will consider moves involving at most s randomly chosen set of

organizations. For this, we first assign to each organization i a weight defined as follows:

$$\omega_i(a^{cur}) = 1 - \frac{|K_i \setminus A_{a^{cur}(i)}|}{|K_i \cup A_{a^{cur}(i)}|}, \tag{12}$$

where $A_{a^{cur}(i)}$ is the set of applications needed by all the organizations $i' \neq i$ installed on cluster $a^{cur}(i)$. The weight function takes positive values and attains value 1 if and only if $K_i \subseteq A_{a^{cur}(i)}$. Note that by moving an organization i to a cluster where many of the applications in K_i are already installed, can lead to a decrease in the installation costs for applications. For this reason, it may be beneficial to move organizations with a low value of $\omega_i(a^{cur})$ to another cluster, and leave the ones with a large value of $\omega_i(a^{cur})$ on $a^{cur}(i)$.

The first step in the neighborhood reduction procedure is to consider moves involving a restricted set of organizations Q. Q is constructed iteratively, starting with $\emptyset$ till it reaches the size s, a preset parameter. In each iteration, an organization from $J \setminus Q$ is chosen to be added to Q with probability $p_i = 1 - \frac{\exp(\omega_i(a^{cur}))}{\sum_{i \in J \setminus Q} \exp(\omega_i(a^{cur}))}$. Observe that since p_i is decreasing in $\omega_i(a^{cur})$, Q will mainly contain organizations for which the set of needed applications does not have a large overlap with the set of applications needed by other organizations on the cluster. The reduced neighborhood is the set of allocations obtained by moves involving only elements in Q.

5. Computational Experiments

In this section, we test the performance of both the IP model and the Tabu Search algorithm on the SaaSCP.

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 together with the LP/IP solver of CPLEX 11.0.1 with default settings. The TS algorithm was programmed in MATLAB R2007b.

5.1. Test Cases

Since there were no prior test cases available in literature, we have generated 32 cases to test our IP model and the Tabu Search heuristic.

In each case, the organizations are of 3 types: large, moderate and small, depending on the number of applications needed and the demand for a certain application. Large organizations require 10, 11 or 12 applications (with equal probability) and the demand for each application is uniformly distributed on $[8000, 10000]$. Moderate organizations require 7, 8 or 9 applications (with equal probability) and the demand per application is uniformly distributed on $[5000, 7000]$. Small organizations require 4, 5 or 6 applications, with a demand per application uniformly distributed on $[2000, 4000]$.

Each organization could choose from a total of 50 applications, divided into five equally sized groups, namely common use, large-specific, moderate-specific, small-specific and the rest. The common use applications have installation costs (in €) uniformly distributed on $[80, 120]$ and each organization chooses 1/3 of

Case	1	2	3	4	5	6
No symmetry breaking						
Time (s)	5.3	7.0	14.1	3.8	5.0	5.2
Nr. Iterations	67,454	96,106	362,719	28,168	68,874	64,169
BB Nodes	2,100	3,132	61,122	1,503	2,605	2,360
Symmetry breaking						
Time (s)	1.1	2.3	2.2	1.6	3.1	3.3
Nr. Iterations	3,123	30,815	32,327	4,012	41,663	29,557
BB Nodes	233	1,937	1,517	300	2,537	1,616

Table 1: IP results cases 1-6 with and without symmetry breaking constraints.

its applications of this group. Depending on its size, each organization requires another 1/3 of its applications from the large, moderate or small-specific groups. The installation costs are uniformly distributed on $[160, 200]$, $[110, 150]$ and on $[60, 100]$ for the large, moderate and small-specific applications respectively. The remainder of applications is randomly chosen with equal probabilities from the rest and the other specific applications groups. For example, for a large organization this means that the remainder was chosen from the rest, moderate- and small-specific application groups.

With the exception of Section 5.3.6, the number of available clusters is equal to the number of organizations. This assumption is very reasonable in practice, where the costs of opening new clusters are much lower than the profit obtained by offering software to an organization. We test however, in Section 5.3.6, the behavior of the algorithm when this is not the case.

The capacity of the clusters is chosen equal to $q_l = \sum_{i=1}^{l} \frac{10000}{1+0.05(i-1)}$ with $l = 1, ..., 20$. In practice, l can be the number of nodes installed on the cluster and it is assumed that the capacity of the cluster is concave in the number of nodes. The costs (in €) of opening a cluster of capacity q_l are set to $1000 + 500l$.

The first 20 small cases contain 10 organizations: 2 large, 5 medium and 3 small. These cases, as we will show later, easily solved to optimality by CPLEX and are mainly used to assess the reduction in computational time when symmetry constraints are added and the quality of the solution obtained by Tabu Search algorithm.

The other 12 large cases (denoted by A-L) consider 60 organizations split into 20 large, 20 moderate and 20 small for cases A-D; 30 large, 10 moderate and 20 small for cases E-H and 10 large, 20 moderate and 30 small for cases I-L. We expect that with test cases it is more difficult for both CPLEX and TS to find good solutions due to larger problem sizes and potentially more overlap of applications when more organizations are involved. As we will demonstrate soon, these cases could not be solved to optimality within reasonable time by CPLEX. They show that TS is more effective in obtaining high quality solutions.

Case	A	D	E	H	I	L
No symmetry breaking						
Best IP Solution	178,050	161,980	199,000	192,170	172,330	141,820
Gap (%)	17.6	16.1	17.05	15.24	17.53	17.15
Nr. Iterations	1,379,116	5,448,337	4,673,554	5,001,367	4,660,600	5,833,457
BB Nodes	2,456	8,528	7,198	7,035	6,131	9,925
Avg. Nr. It. per Node	562	639	649	711	760	588
Symmetry breaking						
Best IP Solution	176,220	161,340	196,520	194,720	174,590	138,610
Gap (%)	13.1	11.8	12.68	12.60	15.98	11.89
Nr. Iterations	6,774,587	7,212,389	7,765,145	6,197,424	2,148,210	6,938,630
BB Nodes	8,018	11,166	10,067	6,234	2,173	8,245
Avg. Nr. It. per Node	845	646	771	994	989	842

Table 2: IP results large cases after two hours with and without symmetry breaking constraints.

5.2. Performance of the IP Model

5.2.1. Small cases

To evaluate both the performance of the IP model and the effect of symmetry breaking constraints given by Equation (7), all the 20 small cases were solved with and without these constraints. Table 1 presents the results for the first 6 cases. We omit the results for the other cases since they have a similar pattern.

Besides the computational time in seconds, we report the number of iterations and nodes (BB nodes) visited in the branch-and-bound tree till the optimal solution is found. In all the cases the use of the symmetry breaking constraints resulted in shorter computation time. This behavior can be explained by the smaller branch and bound tree caused by excluding duplicate nodes that correspond to identical solutions (up to a numbering of the clusters).

5.2.2. Large cases

We further tested the performance of the symmetry breaking constraints on the large cases A-L. Even when the symmetry breaking constraints were added, these cases could not be solved to optimality within 2 hours. In Table 2 we present for 6 large cases the best IP solution found within 2 hours, together with the relative optimality gap, the total number of iterations and the number of nodes in the branch and bound tree. The relative optimality gap is calculated as follows: $\frac{\text{Best IP-LB}}{e^{-10}+\text{Best IP}}$, where Best IP represents the best IP solution value and LB is the best lowerbound found by CPLEX.

As we can see, the symmetry breaking constraints have a positive impact on the IP performance. For each case, the best IP solution improved by 1% when the symmetry breaking constraints are used. In the same time, the relative optimality gap reduced, implying that in the studied cases, the symmetry breaking constraints lead to better lowerbounds.

Note that the average number of iterations per node increased when symmetry breaking constraints were

	Iterations	50	100	150	200	250
No Neigh. Reduction						
	Opt (%)	62.50	97.50	98.50	98.50	98.50
	Opt < 1% (%)	86.00	100.00	100.00	100.00	100.00
	AVG Gap (%)	0.42	0.01	0.00	0.00	0.00
	Max Worst Gap (%)	5.04	0.58	0.09	0.09	0.09
	AVG Time (s)	1.16	2.33	3.49	4.65	5.81
50% Neigh. Reduction						
	Opt (%)	51.50	91.00	96.50	96.50	98.00
	Opt < 1% (%)	87.00	99.50	100.00	100.00	100.00
	AVG Gap (%)	0.39	0.03	0.00	0.00	0.00
	Max Worst Gap (%)	4.13	1.15	0.23	0.23	0.09
	AVG Time (s)	0.80	1.60	2.40	3.21	4.01

Table 3: Results of the TS algorithm with and without neighborhood reduction strategy on the small cases.

used, indicating that for large cases there is a trade-off between the number of nodes in the branch and bound tree and the number of iterations needed to regain optimality. For a time limit of two hours, the overall results for the IP formulation with symmetry breaking constraints were better than the results for the IP formulation without these constraints.

In conclusion, although symmetry breaking constraints resulted in improved results for the SaaSCP, for the large cases, an optimal solution could still not be found within 2 hours. This makes a fast heuristic attractive to use.

5.3. Performance of the Tabu Search Algorithm

5.3.1. Default parameter settings

For every data instance, the TS algorithm was run 10 times with different random initial feasible solutions. The feasible solutions were found by assigning first each organization to a cluster and then randomly performing 10 shift moves. The default parameter settings were chosen as follows. The number T of iterations and the size α of TL_1 were set to $T = 250$ and $\alpha = 10$ for the small cases 1-20, and to $T = 1000$ and $\alpha = 30$ for the large cases A-L respectively. If the neighborhood reduction strategy was used, the neighborhood of the current solution was reduced by 50% per iteration. The intensity of the diversification was set to $\lambda = 0.015$, similar to Cordeau and Laporte (2003). The intensification strategy was started after $\tau = 50$ of non-improving iterations. Finally, the initial penalty for infeasibility was set to $\beta = 1000$ and increased or decreased by a factor of $\theta = 0.1$. The value of the parameters was chosen equal to the value that resulted in the best performance in a series of computational tests. The results of these tests are omitted and only the effect of the neighborhood reduction is further discussed next.

5.3.2. Effect of the neighborhood reduction strategy

We investigate whether the neighborhood reduction strategy can reduce the computational time of the algorithm without compromising on the quality of the solution.

Table 3 shows, for the 20 small cases, the average results obtained with and without the neighborhood reduction strategy incorporated in the Tabu Search. In the row "Opt (%)" the percentage of runs that found the optimal solution is reported. After 50 iterations, the TS without neighborhood reduction found in 62.5% of the runs the optimal solution, while the TS with neighborhood reduction only found the optimum in 51.5% of the runs. As expected, the experiments indicate that it is more effective to consider the whole neighborhood than to reduce the neighborhood. However, after 250 iterations the difference between the two strategies is negligible, suggesting that the benefits of using the whole neighborhood are higher if it is incorporated in the beginning of the search procedure.

The row "Opt $<$ 1% (%)" shows the percentage of the runs with a solution value within 1% of the optimal value. There is no clear difference between the two strategies and after 100 iterations all the runs found a solution with a value close to the optimum. In "AVG Gap (%)" the average performance gap is presented, which is the percentage difference between the solution value generated by one run and the optimal solution value, computed for each run and then average across all the runs for each case. The average gap decreases very fast for both strategies. After 50 iterations the average gap is around 0.42% for TS without neighborhood reduction and 0.39% for TS with neighborhood reduction. After more than 100 iterations the average gap becomes negligible. The row "Max Worst Gap (%)" reports the maximum performance gap in percentage over all the runs. For TS with neighborhood reduction the gap is slightly smaller than for TS with no neighborhood reduction after 50 iterations, but for both the maximum worst gap becomes negligible after 100 iterations.

The advantage of the neighborhood reduction strategy is most evident in the reduction of the computational time, shown in the row "AVG Time (s)". The 50% reduction of the neighborhood lowered the running time of the algorithm by 30% . This suggests that evaluating the neighborhoods is time consuming and that a reduction strategy will be especially useful for larger sized problems with large neighborhoods.

These results indicate that the TS with neighborhood reduction obtains comparable solutions with the TS without neighborhood reduction, in a substantially lower running time. Therefore, we incorporate the neighborhood reduction in our experiments for both small and large data cases.

5.3.3. Small cases

Table 4 presents the results of the TS algorithm with default settings and with 50% neighborhood reduction for the small cases 1-20. For each instance, the exact optimal value is reported in the column "Opt". Column "Nr. Opt" contains the number of runs of the TS that found the optimal solution among 10 runs with different initial solutions, while column "Nr. Opt $<$ 1%" contains the number of runs that resulted in a solution value within 1% of the optimal value. The TS was able to find the optimal value with all different initial allocations for all the cases except case 5, where the solutions found were within 1% of the optimal value. These results demonstrate that the TS algorithm was insensitive with respect to the random

Case	Nr. Opt	Nr. Opt < 1%	Opt	μ (Opt Iter)	Time (s)
1	10	10	22,040	39	1.3
2	10	10	28,100	26	1.3
3	10	10	21,100	80	1.3
4	10	10	20,790	75	1.3
5	6	10	22,070	109	1.3
6	10	10	22,810	59	1.3
7	10	10	28,510	63	1.3
8	10	10	20,190	69	1.3
9	10	10	24,950	64	1.3
10	10	10	27,530	34	1.3
11	10	10	17,730	45	1.3
12	10	10	20,590	62	1.3
13	10	10	21,010	17	1.2
14	10	10	23,430	13	1.3
15	10	10	31,250	9	1.6
16	10	10	27,180	42	1.5
17	10	10	26,040	31	1.6
18	10	10	21,390	81	1.6
19	10	10	27,680	18	1.6
20	10	10	31,080	32	1.6

Table 4: Results of the Tabu search algorithm for cases 1-20.

starting solution.

The column "μ (Opt Iter)" shows the average number of iterations needed in order to find the optimal value. A small μ (Opt Iter) indicates that the optimum solution was found very fast, without employing the diversification and intensification strategies. However, if μ (Opt Iter) is higher than 75, an intensification round was likely started and pushed the search in the direction of the optimal solution. For instance, case 5 resulted in the highest value of μ (Opt Iter), which indicates that there were several local optimums that were hard to pass with the default parameters for the diversification and intensification strategies. This also explains why in case 5, several runs did not find the optimal value. Finally, the last column shows the computational time per run, which was between 1-2 seconds for every case.

5.3.4. Large cases

Next we tested the algorithm on the large cases A-L, which contained 60 organizations. The parameters in the TS were set to their default values and the 50% neighborhood reduction strategy was employed. We compared the results of the TS algorithm with the best solution obtained within 2 hours by the IP-model with symmetry breaking constraints. As we have seen in Table 2, the IP solutions we used for comparison are not optimal, the relative optimality gap being between 11%-16%.

Table 5 contains the results of these experiments. Column "Nr. Best" reports the number of times the best solution was found for the 10 different runs. The average number of runs in which TS found the same

Case	Runs	Nr. Best	Nr. Best $< 1\%$	BEST	IP Improv (%)	Time (s)
A	10	3	10	167,930	4.70	216.3
B	10	7	10	173,560	4.15	221.7
C	10	3	10	165,030	3.78	214.3
D	10	2	10	153,540	4.83	389.0
E	10	7	10	190,010	3.31	252.4
F	10	9	10	186,430	3.76	255.9
G	10	6	10	190,690	2.24	256.2
H	10	7	10	185,480	3.48	264.7
I	10	9	10	162,840	6.73	238.3
J	10	10	10	151,230	5.79	206.2
K	10	3	10	172,010	5.76	276.8
L	10	7	10	132,670	4.29	239.5

Table 5: Results of the Tabu search algorithm for cases A-L.

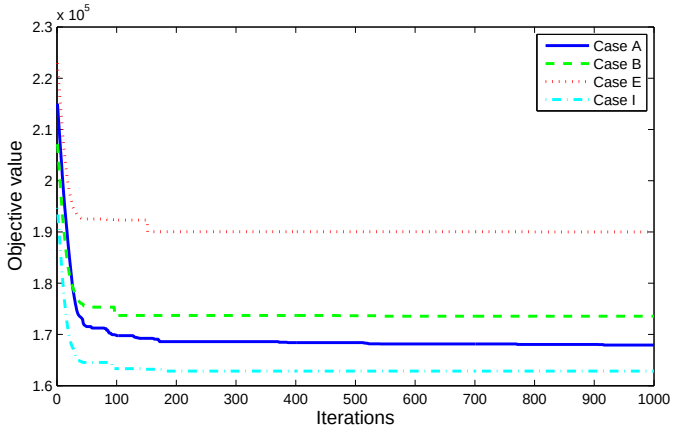
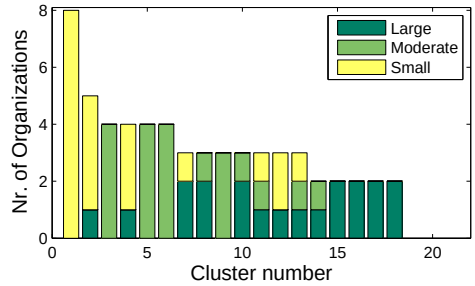


Figure 2: Convergence results TS for large cases A, B, E and I.

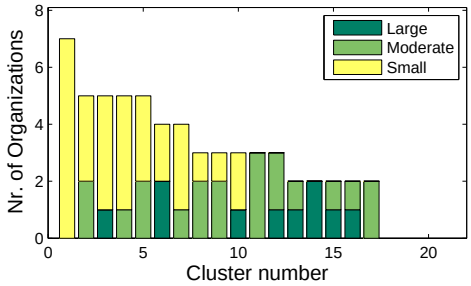
solution is 6, the minimum being 2 for case D and maximum 10 for case J. However, as column "Nr. Best $< 1\%$" shows, even if the solutions found differ, they are all within 1% of the best known solution value.

In the column "IP Improv (%)", the improvement (in percentages) on the best known IP value obtained by CPLEX is presented. As we can see, TS found always a better solution value than the best solution value found by CPLEX. The average improvement was around 4% for all the cases. In the last column the computational time per run is shown. In each case, the running time was around 3 to 4 minutes, significantly less than the 2 hours time limit of CPLEX.

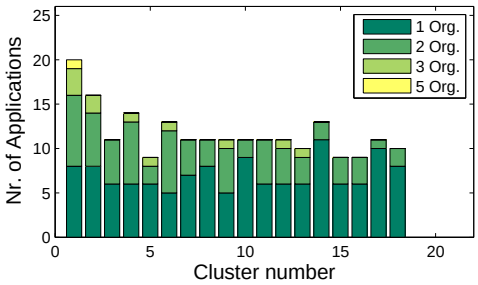
Finally, Figure 2 shows the convergence results for 4 large cases, namely A, B, E and I. In the figure the objective value is plotted against the number of iterations for each of the four cases. The results illustrate the quick convergence behavior of the proposed TS. Within only 200 iterations all the cases have converged to their best known solution values. In the remaining iterations the solutions improve only slightly.



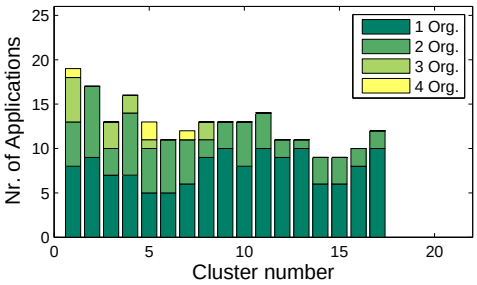
(a) The number of organizations per cluster (Case A).



(b) The number of organizations per cluster (Case I).



(c) Overlap in applications (Case A).



(d) Overlap in applications (Case I).

Figure 3: Structure of the best solution found by TS for case A and I.

5.3.5. On the structure of the clusters in the TS solutions

As suggested by the complexity proof the overlapping applications between organizations play an important role in finding optimal solutions in SaaSCP. Hence in this subsection, we analyze in more detail the solutions obtained in order to gain more insight into the type of organizations that are placed together on a cluster and their overlap in applications. We will limit our discussion to the large cases A and I.

Figures 3a and 3b show in the obtained solutions the total number of organizations per cluster and their types: small, moderate, large. Figures 3c and 3d present the total number of applications installed in a cluster and the number of organizations an application is installed for. 18 and 17 clusters were used according to the resulting allocations with cases A and I respectively. For Case A, 8 small organizations requesting a total of 20 applications were installed on the first cluster. Of those 20 applications, 8 applications were installed for one organization, 8 for two, 3 for three and 1 for five organizations. As a comparison, on the cluster 18, 2 organizations were assigned and in total 10 applications were installed. Of these 10 applications, only 2 are overlapping applications, and the remaining 8 were installed only for one organization.

The similar pattern can also be found for Case I. These results suggest that it is beneficial to install on a cluster organizations with a large overlap in applications. Furthermore, a very intuitive result which is confirmed by the results is that there is more overlap in applications on clusters with more organizations.

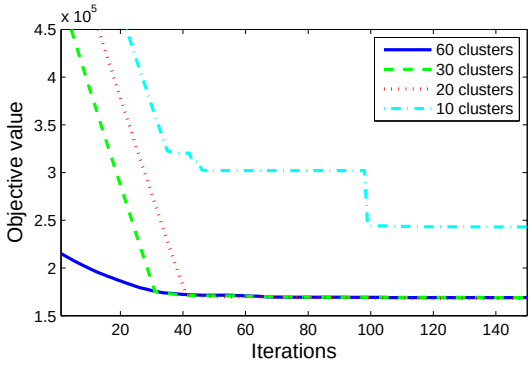


Figure 4: Convergence results TS when starting with an infeasible starting solution.

5.3.6. Results for cases with limited number of clusters

In all the results reported so far we have assumed that the number of available clusters is equal to the number of organizations, which makes the problem of finding an initial feasible solution straightforward. One may wonder, however, how to find such a feasible solution if the number of clusters is strictly smaller than the number of the organizations. In this case, it is not easy to find an initial, feasible allocation, as finding a feasible solution is NP-hard. In Remark 1 we have suggested to add dummy clusters of arbitrary high costs and to run the TS on the new instance.

We now investigate the results obtained by using this heuristic on test case A where there are 60 organizations. We varied the number of available clusters from 30, 20 to 10. In all these cases dummy clusters are added in order to create 60 clusters in total. The cost of a dummy cluster is chosen to be 10 times the original cost. We also included the case with 60 clusters for the purpose of comparison.

In Figure 4 it can be seen that the cases with 20 and 30 clusters quickly converge to the same solution value as the one achieved by 60 clusters. Although starting from infeasible initial solutions (with high costs of more than €4.5×10^5), for the cases with 30 and 20 clusters, within the first 30 and 40 iterations, the TS will shift the organizations from the dummy clusters to the non-dummy clusters, and will find a feasible solution. These results show that the heuristic of adding dummy clusters performs very well in terms of the convergence speed and the quality of the solution found by the TS.

When there are only 10 clusters available, however, the TS returns no feasible solution. This can be explained by the fact that the total capacity of 10 clusters is about 1.4×10^6, which is lower than the total demand of all organizations (1.9×10^6). Thus there is no feasible solution with 10 clusters.

6. Conclusion

In this article we have addressed the problem of designing a Software as Service network of minimal costs. We have proved that the problem is NP-hard, by establishing relationships with the graph partitioning problem. Subsequently, we have presented an Integer Programming model and proposed a Tabu

Search algorithm for finding feasible solutions. We designed a set of problem cases and conducted extensive experiments to test the solution quality and the running time of the TS algorithm. The results obtained by the algorithm were compared with results obtained by solving the IP model with CPLEX. For small problem instances, TS could find in 95% of the cases the optimal solution, and in the other 5% it found a solution with a value within 1% of the optimal value. For large problem instances, TS found in 3-4 minutes solutions of better quality than CPLEX after 2 hours. From the experimental results, we expect that the proposed Tabu search heuristic is able to return good solutions within a reasonable running time for even larger problem instances, especially if it is implemented with a more efficient programming language such as C/C++ or Java.

In this paper we developed heuristics for solving the clustering problem in SaaS. A related question is whether there exist approximation algorithms that can guarantee some performance bound. This is not trivial since as we discussed in Section 2, the SaaSCP has characteristics of several NP-hard problems.

Another interesting future work is to investigate SaaSCP in game theoretical settings. For instance, from the software offering company's point of view, one could be interested how the total cost should be distributed among organizations such that it is fair and in the core. On the other hand, from the organization's point of view, what are good strategies in revealing its request such that its cost can be minimized.

References

Ahuja, R., Orlin, J., Pallottino, S., Scaparra, M., Scutella, M., 2004. A Multi-Exchange Heuristic for The Single-Source Capacitated Facility Location Problem. Management Science 50 (6), 749–760.

Barahona, F., Chudak, F., 2005. Near-optimal solutions to large-scale facility location problems. Discrete Optimization 2 (1), 35–50.

Beasley, J., 1993. Lagrangean heuristics for location problems. European Journal of Operational Research 65 (3), 383 – 399.

Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., Munro, M., 2000. Service-based software: the future for flexible software. Asia-Pacific Software Engineering Conference 0, 214.

Cordeau, J., Laporte, G., 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological 37 (6), 579–594.

Crainic, T., Perboli, G., Rei, W., Tadei, R., 2011. Efficient lower bounds and heuristics for the variable cost and size bin packing problem, In Press, Accepted Manuscript.

Crainic, T., Perboli, G., Tadei, R., 2009. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. European Journal of Operational Research 195 (3), 744 – 760.

Delmaire, H., Diaz, J., Fernandez, E., Ortega, M., 1999. Reactive grasp and tabu search based heuristics for the single source capacitated plant location problem. INFOR, Canadian Journal of Operational Research and Information Processing 37, 194–225.

Garey, M., Johnson, D., 1979. Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. WH Freeman and Company, San Francisco, Calif.

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

Ghosh, D., 2003. Neighborhood search heuristics for the uncapacitated facility location problem. European Journal of Operational Research 150 (1), 150 – 162.

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

Holmberg, K., Rónnqvist, M., Yuan, D., 1999. An exact algorithm for the capacitated facility location problems with single sourcing. European Journal of Operational Research 113, 544–559.

Jans, R., 2009. Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. INFORMS Journal on Computing 21 (1), 123–136.

Kang, J., Park, S., 2003. Algorithms for the variable sized bin packing problem. European Journal of Operational Research 147 (2), 365–372.

Kernighan, B., Lin, S., 1970. An efficient heuristic procedure for partitioning graphs. Bell Systems Journal 49, 291–307.

Lee, C., 1991. An optimal algorithm for the multiproduct capacitated facility location problem with a choice of facility type. Computers and Operations Research 18, 167–182.

Lodi, A., Martello, S., Vigo, D., 1999. Approximation algorithms for the oriented two-dimensional bin packing problem. European Journal of Operational Research 112 (1), 158 – 166.

Mazzola, J., Neeb, e. A., 1999. Lagrangian-relaxation-based solution procedures for a multiproduct capacitated facility location problem. European Journal of Operational Research 115, 285–299.

Melo, M., Nickel, S., Saldanha da Gama, F., 2005. Dynammic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. Computers and Operations Research 33, 181–208.

Mertz, S. A., Eschinger, C., Eid, T., Swinehart, H. H., Pang, C., Wurster, L. F., Dharmasthira, Y., Pring, B., 2010. Forecast analysis: Software as a service, worldwide, 2009-2014, update. Tech. rep., Gartner.

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

Neebe, A., Rao, M., 1980. An algorithm for the fixed-charge assign- ing users to sources problem. Journal of Operational Research Society 34, 1107–1113.

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.

Pirkul, H., 1987. Efficient algorithms for the capacitated concentrator location problem. Computers & OR 14 (3), 197–208.

Pirkul, H., Jayarama, V., 1998. A multi-commodity, multi-plant, capacitated facility location problem: Formulation and efficient heuristic solution. Computers & OR 25 (10), 869–878.

Rönnqvist, M., Tragantalerngsak, S., Hol, T. J., 1999. A repeated matching heuristic for the single-source capacitated facility location problem. European Journal of Operational Research 116 (1), 51–68.

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.

Sääksjärvi, M., Lassila, A., Nordström, H., 2005. Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing. In: Proceedings of the IADIS International Conference e-Society 2005. pp. 177–186.

Sherali, H., Smith, J., 2001. Improving discrete model representations via symmetry considerations. Management Science 47 (10), 1396–1407.

Sridharan, R., 1993. A Lagrangian heuristic for the capacitated plant location problem with single source constraints. European Journal of Operational Research 66, 305–312.

Sridharan, R., 1995. The Capacitated Plant Location Problem. European Journal of Operational Research 87, 203–213.

Sun, M., 2006. Solving the uncapacitated facility location problem using tabu search. Computers & Operations Research 33 (9), 2563–2589.

Xin, M., Levina, N., 2008. Software-as-a-Service Model: Elaborating Client-Side Adoption Factors. In: Proceedings of the 29th International Conference on Information Systems.

Publications in the Report Series Research* in Management

ERIM Research Program: "Business Processes, Logistics and Information Systems"

2011

Sequencing Heuristics for Storing and Retrieving Unit Loads in 3D Compact Automated Warehousing Systems
Yugang Yu and René B.M. De Koster
ERS-2011-003-LIS
http://hdl.handle.net/1765/22722

A Local Search Algorithm for Clustering in Software as a Service Networks
Jelmer P. van der Gaast, Cornelius A. Rietveld, Adriana F. Gabor, and Yingqian Zhang
ERS-2011-004-LIS
http://hdl.handle.net/1765/22723

---

* A complete overview of the ERIM Report Series Research in Management:
https://ep.eur.nl/handle/1765/1

ERIM Research Programs:
LIS Business Processes, Logistics and Information Systems
ORG Organizing for Performance
MKT Marketing
F&A Finance and Accounting
STR Strategy and Entrepreneurship