Computers & Operations Research 50 (2014) 38–46

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A Tabu Search Algorithm for application placement in computer clustering

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

ARTICLE INFO

Available online 18 April 2014

Keywords:
Placement problem
Facility location
Generalized binpacking
Tabu search
Integer programming

ABSTRACT

This paper presents and analyzes a model for the problem of placing applications on computer clusters (APP). In this problem, organizations requesting a set of software applications have to be assigned to computer clusters such that the costs of opening clusters and installing the necessary applications are minimized. This problem is related to known OR problems such as the multiproduct facility location problem and the generalized bin packing problem. We show that APP is NP-hard, and then propose a simple Tabu Search heuristic to solve it. The performance of the Tabu Search heuristic is assessed via extensive computational experiments, which indicate the promise of the proposed Tabu Search.

1. Introduction

Since the early '90s there has been a rapid advance in the Internet in terms of speed, connectivity and reliability. The fact that the Internet has become virtually available everywhere, coupled with the interest of vendors in capturing the market of small customers who cannot easily afford expensive enterprise software, has fueled the rise of *Software as a Service* (SaaS) [1]. Companies that offer SaaS manage and maintain their software centrally, and offer the software to customers via Internet. In this way, the costs for purchasing the software, including maintaining and upgrading it, are shared among several customers. At the same time, customers have realized that good backup and fault-tolerant practices have made data more secure and reliable with the SaaS offering company rather than with their own enterprise. Especially in recent years, Cloud computing has become an emerging computing paradigm [2,3]. As one of the major services provided over Cloud computing, the market for SaaS is rapidly growing. Examples of the key SaaS providers include IBM, Microsoft, Oracle, and SalesForce.com.

This paper focuses on the following application placement problem (APP) encountered in computer clustering in SaaS networks [4]. The input of APP is a set of organizations, a list of software applications desired by each organization, together with the resource requirements (demand) associated with each organization and its desired software application. A set of choices for computer clusters of different capacities (transactions/second) is also given. The goal of APP is to decide how many computer clusters (and of which capacity) to open, and which organizations to install on each of them such that the costs are minimized. Costs are incurred when a new cluster is opened and when applications are installed on a cluster. An important requirement in APP is that if an organization is assigned to a cluster, then all the applications desired by this organization should be installed on the cluster as well.

The APP 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, the input is a set of sites where facilities of different capacities may be opened, as well as a set of client locations, together with a list of desired products per client. The goal is to decide where to open facilities, what products to manufacture at each location and how to allocate clients to the open facilities, such that the total costs (for opening facilities, installing specific equipment for manufacturing and transportation costs) are minimized. The APP can be viewed as an MPCFL as follows: the clusters that have to be opened can be viewed as facilities, the organizations as clients and the applications as the products desired by clients. Since there are no costs for assigning organizations to clusters, the transportation costs in the corresponding MPCFL are zero. Note that since one of the requirements of APP is to assign each organization to one cluster, in the corresponding MPCFL each client should be served by only one facility. To the best of our knowledge, this variant of MPCFL has not received much attention in the OR literature. However, several papers address the variant in which a customer may be served by several facilities. Lee solved this problem via

* Corresponding author.
E-mail addresses: jgaast@rsm.nl (J.P. van der Gaast), nrietveld@ese.eur.nl (C.A. Rietveld), gabor@ese.eur.nl (A.F. Gabor), yqzhang@ese.eur.nl (Y. Zhang).

http://dx.doi.org/10.1016/j.cor.2014.03.025
0305-0548/© 2014 Elsevier Ltd. All rights reserved.

Bender's decomposition [5] and via a combination of Bender's decomposition and Lagrangean relaxation [6]. Mazzola and Neebe [7] developed Lagrangean heuristics for the variant in which facilities of different types may be opened at a location. A generalization of this problem in the context of supply chain design is also studied in [8].

Another strongly related problem to APP is the generalized bin packing problem (GBPP). In this problem, a set of objects has to be assigned to a set of bins of different volumes. Costs are incurred when a bin is used. It is easy to see that the GBPP is a special case of the APP, when all the organizations require the same applications. In this case, an organization corresponds to an object in GBPP, the demand of an organization corresponds to the volume of the object and the clusters correspond to bins. Having the bin packing problem as a special case, GBPP is NP-hard. The worst-case behavior of two greedy algorithms for variants of GBPP in which the cost of unit size of each bin does not increase when the volume of the bin increases is analyzed in [9]. Both algorithms are optimal if the volumes of objects and bins are divisible; that is, for any two objects (bins), the volume of the largest is divisible by the volume of the smallest. If only the volumes of the bins are divisible, the algorithms give a solution whose value is less than $\frac{11}{9}C(B^*)+4\frac{11}{9}$, where $C(B^*)$ is the value of the optimal solution. For the variant with non-divisible bin sizes, the solution is within $\frac{3}{2}C(B^*)+1$. In [10], the authors present extensive numerical studies of several heuristics inspired from known heuristics for the classical bin packing and knapsack problems.

The Tenant Placement Problem (TPP) is also a related problem to APP. TPP concerns the placement of tenants (or organizations) on a set of available servers at minimal cost. In [12], the authors present a greedy algorithm for the online optimal placement of tenants and applications. A similar online tenant placement problem is studied in [13], which aims to maximize the total number of tenants (organizations) whose applications are installed on a set of servers of limited capacity (in their case, CPU or space). The authors propose several heuristics and compare their behavior via extensive numerical experiments. The main differences between our problem and theirs [12,13] are the following: firstly, we consider offline (initial) tenant placement, and secondly, we assume that the applications corresponding to each tenant should be installed on one server. In [14], the focus is on the problem of finding an offline and online placement of applications on a set of servers with limited capacity (CPU or network bandwidth) such that the number of installed applications is maximized (max-APP). Each application may contain several components, with known resource requirements, that can be placed on different servers. Components cannot be shared among applications, but components of the same application may be installed on different nodes. This problem can be reduced to the bin packing problem and is thus NP-complete. The authors propose 2-approximation algorithms for solving some variants of max-APP. Besides the different optimization objectives, our problem differs from the max-APP in the following ways: in max-APP (i) the capacity of a server is given, (ii) there is no cost of placing an application, and (iii) a component cannot be used by several applications, whereas in APP organizations may share the same applications.

The PMAX problem analyzes the optimal placement of organizations on servers, while taking into account the costs of the servers and penalty costs for not fulfilling the service level agreements of the organizations [19]. The load of the servers is shared among organizations. Two approximation algorithms for this problem are presented. The main difference between APP and PMAX is that in APP, we do consider not only the costs of servers, but also the costs and placement of applications. Due to the overlap in applications, the placement of applications is not trivial.

1.1. Contribution

We summarize the contributions of our work as follows:

- The APP is a highly relevant problem within the SaaS domain, and it combines features characteristic of several NP-hard problems (facility location and generalized bin packing problem). This makes the problem itself theoretically interesting to investigate.
- We build a formal model for solving APP and show that the problem is NP-hard. Furthermore, we propose a Tabu Search heuristic that is able to rapidly find good solutions for many problem instances.
- By conducting a set of experiments with different settings, we provide some insights on the difficulty of solving this problem, and thus indicate a starting point for further investigation of good approximations and heuristics.

1.2. Organization of the paper

The paper is organized as follows. Section 2 provides a formal description of the problem and studies its the computational complexity. Section 3 proposes an integer linear program. 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 indicates some possible further research topics.

2. Formal problem description and computational complexity

2.1. Formal problem description

The application placement problem (APP) 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 $i \in I$, a list $K_i \subseteq K$ of applications needed and the demand d_{ik} for each application $k \in K_i$ are given. The demand of an organization for an application is defined as the resource requirement (transactions/second) associated with the organization and application. Demand usually depends on the number of users the organization has and the characteristics of the application.

Clusters have different capacities (transactions/second) 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 is *feasible* if (1) for every organization $i \in I$, all its required applications are installed on 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 that cluster. Otherwise an allocation is called as *infeasible*.

The goal in the APP is to decide on 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.

2.2. Computational complexity

When all organizations need the same set of applications, APP reduces to a generalized bin packing problem; it is thus NP-complete. This reduction, however, does not provide sufficient insight into the computational difficulty of the APP, since in practice it rarely occurs that all organizations require exactly the same applications. In order to understand the role played by the overlap of the lists of different organizations, we give an alternative proof for its NP-completeness based on a reduction from the graph partitioning problem (*GP*).

Graph partitioning [11]: 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 $V_1, V_2, \ldots, V_n$ such that $\sum_{v\in V_i} w(v)\le O$ for $1\le i\le n$ and such that if $\hat{E}\subseteq E$ is the set of edges that have their two endpoints in two different sets, $\sum_{e\in\hat{E}} l(e)\le R$?

We now show that the APP problem is NP-complete.

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

Proof. First we show that the problem is in NP. Given a problem instance of the APP 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 APP is NP-hard by reducing from GP.

Given a GP problem with graph $G=(V,E)$ and integers O and R, we construct an APP 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 Fig. 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 APP 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 $\sum_{v\in V_i} w(v)\le O$ and $\sum_{e\in\hat{E}} l(e)\le R$, where $\hat{E}$ is the set of cut edges that have their two endpoints in two sets. Then in the constructed APP 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 $\sum_{v\in V_j} w(v)\le O$. This implies $\sum_{v'\in V'_j} D_{v'}=\sum_{v\in V_j} w(v)\le O$; 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 need therefore to be installed only once. Since there is no cost of openingclusters, the cost of allocation $C(a)$ is simply the total number of applications that need to be installed (i.e., $C(a)=|E'|+\sum_{e\in\hat{E}} l(e)\le |E'|+R=X$).

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, and total of cost $C(a)\le X$ in the corresponding GP problem), there exists a partitioning of V to n sets, and the total weight of the cut edges is $\sum_{e\in\hat{E}} l(e)=C(a)-|E'|\le X-|E'|=R$.

Therefore, APP is NP-complete. □

Since the problem is NP-complete, there exists no polynomial time algorithm for finding an optimal solution, unless P=NP. Before proposing a heuristic to solve the APP, we give an Integer Programming formulation, which will be used in our computational experiments to assess the quality of the heuristic.

3. An integer programming formulation

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

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

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

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

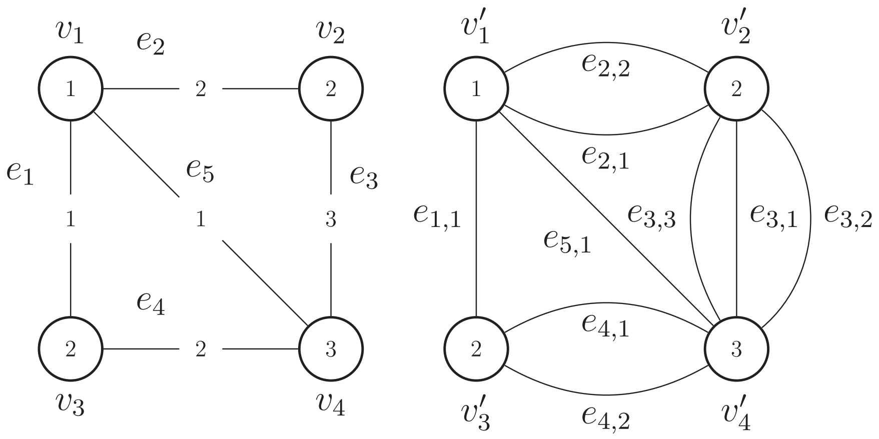

Fig. 1. Graphical example of the reduction from GP to APP. The optimal solution for the GP instance (with $O=6$) is the partition $V_1=\{v_3\}$, $V_2=\{v_1, v_2, v_4\}$, with the minimal edge cuts 3. The figure on the right-hand side is the constructed APP instance, where given the cluster's maximal 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.

The APP problem can be modeled as an (IP) as follows:

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

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

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

$$x_{ij} \le z_{kj}, \quad k\in K_i,\ i\in I,\ j\in J \tag{4}$$

$$\sum_{l\in L} v_{jl} \le 1, \quad j\in J \tag{5}$$

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

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 i is assigned to exactly one cluster. Constraints (3) state that a cluster cannot serve more than the installed capacity. These also imply that one cluster should be opened if at least one 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.

3.1. Reducing symmetry and strengthening the LP relaxation

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, there are multiple feasible solutions with the same objective value, differing only in the numbering of the clusters, but not in the set of organizations assigned to the same cluster. In order to reduce symmetry, one may add symmetry-breaking constraints; see, for example, [15,16].

For APP, we first replaced constraint (2) by

$$\sum_{j\le i} x_{ij} = 1,$$

which imposes that each organization i is assigned to a cluster $j\le i$. We assumed that organizations are numbered in the decreasing order of the total costs of the applications needed. Note that this is possible due to the fact that the costs of opening a given cluster depend only on the capacity of the cluster. To reduce symmetry even further, we propose adding constraints that ensure that organization i can be assigned to cluster j only if organizations $1,\dots,i-1$ are assigned to clusters $1,\dots,j-1,j$, in other words, that the vectors $(x_{1j},\dots,x_{mj})$ are lexicographically greater ($\ge_L$) than $(x_{1,j+1},\dots,x_{m,j+1})$ for any cluster $j\ge 1$. This can be done by adding constraints of the form:

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

This constraint is equivalent to $(x_{1j},\dots,x_{mj}) \ge_L (x_{1,j+1},\dots,x_{m,j+1})$, as it is proven in Proposition 2 in [15].

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 [17]. A comprehensive introduction to TS can be found in [18]. This section introduces a Tabu Search heuristic for the APP. We first briefly mention some terminology.

A *move* from allocation a to allocation a' corresponds to the set of subsequent assignments of organizations to clusters, which leads from a to a'. This paper considers 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, the cluster is closed. Also, if there is an application which is no longer requested by an organization, the installation costs for this cluster are correspondingly reduced. In an interchange move, denoted by $\varphi_{int}(a,a')$, two organizations switch the clusters to which they are assigned. Similar to the shift move, the installation costs on the two clusters are updated accordingly after every interchange move. Note that although one interchange move could be obtained by two shift moves, in many situations, using shift moves alone in the TS may not reach same allocations as a result of interchange moves. Observe that a move can be performed even if the allocation obtained exceeds the total capacity of the available clusters.

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. Since a move does not necessarily lead to a feasible allocation, a neighborhood may contain infeasible allocations.

The *Tabu list* maintains a set of forbidden moves, with the primary goal of preventing the algorithm from cycling. It is implemented as a first-in-first-out (FIFO) queue.

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 as Tabu. We use the most commonly used aspiration criterion, which consists of allowing a move if it results in a solution with a better objective value than the current best-known solution.

The search terminates by the *Stopping Criterion*, which is a predefined number of iterations T.

We now present a Tabu Search algorithm, which 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. 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 a client. To find an initial feasible solution, we start with a number of clusters equal to total demand/max. capacity of a cluster. We assign each organization randomly to a cluster that has enough remaining capacity. If there is no cluster with sufficient capacity, we add a new cluster and reassign all organizations. We initialize the current solution a^{cur} and the so-far best solution a^{best} as a^{ini}. The Tabu list TL is at this moment empty, which means that all the moves are allowed.
2. *Improvement phase*: The improvement phase evaluates each allocation a in $\mathcal{N}(a^{cur})$ and finds the allocation with the lowest value.

2a. *Calculate the value of allocations in $\mathcal{N}(a^{cur})$*: The value $v(a)$ of an allocation a is defined as $v(a)=f(a)+p(a)$, where $f(a)$ represents the costs associated with a, and $p(a)$ is a penalty function for exceeding the maximal capacity of the cluster.
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 $R(j) = \arg\min\{l \mid \sum_{i:a(i)=j} \sum_{k \in K_i} d_{ik} \le q_l\}$. Hence,

$$f(a) = \sum_{j \in a(I)} \sum_{k \in \cup_{i:a(i)=j} K_i} c_k + e_{R(j)}. \tag{8}$$

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

$$p(a) = \beta \sum_{j \in J} \left[\sum_{i \in I: a(i)=j} \sum_{k \in K_i} d_{ik} - q_{max} \right]^+ \tag{9}$$

where $[b]^+ = \max(0, b)$. The parameter β is dynamically adjusted during the search in order to explore more intensively the solutions that do not violate the constraints too much. Before calculating $p(a)$, 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 the infeasible solution, while the search is directed towards exploring feasible solutions if β is high. Section 5.2 investigates the effect of the parameters on the TS algorithm.

2b. *Choose 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 by one that is Tabu but satisfies the aspiration criterion chosen. Denote by $\tilde{a}$ the solution found by the neighborhood search.

2c. *Modify the Tabu list*: In this phase new moves are added to the Tabu list and old moves are deleted. The Tabu list TL has a maximal fixed size α. 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 from j to $a^{cur}(i)$ becomes Tabu and is added to TL. 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 $a^{cur}(i)$ from $a^{cur}(i')$, and i' to $a^{cur}(i')$ from $a^{cur}(i)$ 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.

3. *Check the stopping criteria and restart search*: The iteration counter is increased by one after each iteration. If the maximum number T of iterations has not been reached, we proceed with Step 1. If the best solution a^{best} has remained unchanged in the last τ iterations, it may indicate that the search algorithm is stuck in a local optimum. In this case, we restart the Tabu Search algorithm with a randomly generated, feasible initial solution and set the Tabu list empty. This allows the algorithm to explore different regions of the search space. If the maximum number of iterations has been reached, the algorithm is terminated with the current best solution a^{best} as the allocation with the lowest cost for the APP.

5. Computational experiments

This section explains how the test instances are generated, investigates the behavior of the proposed Tabu Search algorithm, and compares its results with the optimal solution obtained by solving the integer programming formulation.

All results reported in this section were obtained on a Core i5 with 1.7 GHz and 4 GB of RAM. The integer program was solved in CPLEX 12.3.0. The TS algorithm was programmed in Java 7, adapted from the OpenTS, a Java tabu search framework.[1]

[1] http://www.coin-or.org/Ots/

5.1. Test instances

Since there were no prior test instances available in the literature, we have generated instances based on realistic parameters obtained from data provided by a SaaS offering company in the Netherlands.

In order to test the influence of the number of required applications and the capacity of the clusters on the solution, we considered in each instance three types of organizations: large, medium and small. Large organizations require 20–30 applications (with equal probability) and the demand for each application is uniformly distributed on [80, 100] transactions per second. Medium-sized organizations require 10–15 applications (with equal probability) and the demand per application is uniformly distributed on [50, 70] transactions per second. Finally, small organizations require 4–6 applications, with a demand per application uniformly distributed on [20, 40] transactions per second.

Further, in order to study the influence of the application overlap on the solution, the applications of each organization were selected from a set of applications, divided into common-use, large-specific, medium-specific and small-specific applications. The common-use applications can be requested regardless of the size of the organization – large, medium or small – while the specific applications can only be requested by the organization of the same type. Each application group has different installation costs which are chosen uniformly. It costs from €750 up to €1000 to install a large-specific application on a cluster, from €300 up to €500 for a medium-specific, from €50 up to from €100 for a small-specific and from €50 up to €1000 for a common-use application.

We assume that the number of available clusters is equal to the number of organizations. The capacity of the clusters is chosen equal to $q_l = \sum_{i=1}^{l} 1000(1-0.05(i-1))$, where $l = 1, \ldots, 10$ is the number of nodes installed on the cluster. In practice, it is assumed that the capacity of the cluster is concave in the number of nodes. The costs of opening a cluster of capacity q_l are set to €$30,000 + 1000l$.

We consider 45 organizations that require applications from a set of 45 applications. We generated 4 cases, A, B, C and D, where each case contains 10 similar generated instances with random demands. The cases differ in the distribution of the number of small, medium and large organizations (see Table 1). In addition, we generated a larger case E, which contains 60 organizations.

5.2. Evaluation of the tabu search algorithm

The purpose of this set of experiments is not to find optimal values for the parameters of the TS on the test instances, but to gauge when the TS provides desirable search behaviors. We evaluate the effect of the following components and parameters in the algorithm: (1) the random re-starting strategy; and (2) the values of θ that determine the penalty value for exceeding the capacity of the cluster (see Eq. (9)). In addition, we evaluate how the initial solutions influence the performance of the TS algorithm.

The parameter settings were chosen as follows. We set the size α of the Tabu list to 500. As the number of possible moves given an allocation is large (i.e., $|I|(|J|-1)+\binom{|I|}{2} = O(|I|^2)$, where $|I|$ is the number of organizations), the size α of the Tabu list should not be too small. The number T of iterations was set to 20,000, which is sufficient for us to see the search behavior of the algorithm. The initial penalty for infeasibility in the penalty function was set to $\beta = 200$ and was increased or decreased by a factor of θ.

We first study the effect of θ that influences the penalty function (Eq. (9)) by decreasing the value of β when a solution is feasible, and by increasing it otherwise. We expect that a very small value of θ will encourage less the search, compared to a relatively higher θ, as for a small value of θ, the algorithm barely takes into account the feasibility of the current solution in deciding the search direction. On the other hand, a very high value of θ may make the search more random. To see its effect better, we ran the TS without the restarting strategy. In this set of experiments, we varied θ from 0.0005 to 5 in order to confirm our expectation. We elected to show here the results on one instance with $\theta=0.5$ and $\theta=0.0005$. From Fig. 2, note that a very small θ (Fig. 2a) leads the search procedure to explore rather restricted areas; consequently, it is more difficult for the TS to find improving solutions in the neighborhood. In this case, the solution quality provided by the TS could heavily depend on the initial search regions that are determined by the initial solutions. For the sake of comparison, with a higher θ, like $\theta=0.5$ (Fig. 2b), the algorithm visits more potentially promising solutions.We prefer the search behavior given by a larger θ, as it is more promising with regards to finding the optimal solution. Therefore we choose the value of θ to 0.5.

To prevent the search becoming stuck in local optima, we proposed re-starting the TS algorithm with a random feasible solution if it could not find an improved solution after τ iterations. Here we investigate the effect of this restarting strategy. The algorithm should not re-start too quickly, as TS needs sufficient iterations to explore the neighborhood solutions. We tested the TS algorithm, both with restarting strategy after $\tau=1000$ non-improving iterations and without restarting strategy, on the 10 instances of case A. The performance differences are significant. We include here two results in order to illustrate the behavior of the TS algorithm.

In Fig. 3, the solution value is plotted against the number of iterations for the instance 2 of case A. Comparison of the two graphs in the figure shows that without re-start, the proposed TS algorithm is not able to jump out of local optima (Fig. 3a). In this case, the restarting strategy works well, as it can lead the search to different solution regions, which facilitates finding an optimal solution (Fig. 3b).

Finally, we test how the initial random allocations influence the performance of the TS algorithm. In this set of experiments, the number of iterations $T=10{,}000$ and we did not use the re-starting strategy. Note that as Fig. 3a suggests, if no re-starting strategy is used, 10,000 iterations are enough for the analysis of the TS algorithm. For each instance of each case, we ran the TS algorithm with randomly generated initial allocations 500 times and record the best solution found in each run. We report here the results on case A and case B. From the results, the average relative difference of performance between the mean and best solutions is 0.74% and 2.92%, for cases A and B respectively. The average relative difference of performance between the worst and best solutions is 10.4% for case A and 5.33% for case B. Furthermore, to have a better idea on the distribution of the solutions, we present the relative errors with respect to the best solutions found in 500 runs in Fig. 4, where for each case, we show the results of such differences for 10 instances together in one graph. In about 7% of the runs for cases A and B, the TS algorithm found the best solutions. For case A, the worst solutions were still within 90% of the best, while for case B, they were within 95%. The difference between the mean behavior and the worst case suggests that the restart procedure improves the quality of the solution considerably. For the restart procedure to be efficient, we suggest to choose the maximal number of iteration T large, such that if a local optimum is found after a small number of iterations, the algorithm has the chance to explore neighborhoods of different initial solutions.

5.3. Performance of the TS algorithm

We test the performance of the IP model and the Tabu Search heuristic in terms of (1) the solution quality, and (2) the running time. We also compare them with a simple greedy and local search heuristic.

In order to confirm the benefits of the symmetry-breaking constraints given by Eq. (7) of the IP model, we first ran a set of exploratory experiments on 120 small problem instances that contain at most 18 organizations. In all the cases, the use of the symmetry-breaking constraints resulted in shorter average computational times, and all instances could be solved to optimality within half an hour using the symmetry-breaking constraints, compared to two hours running time without symmetry breaking. The smaller branch-and-bound tree explains this behavior, because duplicate nodes that correspond to identical solutions (up to a numbering of the clusters) are removed.

We now report the performance of the IP model on the cases A–E in Table 1, with a maximum running time limit of 4 h for each instance. In Table 2 we present the average relative optimality gap, the average total number of iterations and the average number of nodes in the branch-and-bound tree. The relative optimality gap is calculated as follows:

$$Gap = \frac{\text{Best IP solution} - \text{Best Lowerbound}}{e^{-10} + \text{Best IP solution}}$$

where the best IP solution value and the best lower bound are found by CPLEX.

As can be seen from the table, although the IP model was not able to solve the instances to optimality, the average relative gaps

Table 1
The five test cases.

Case	A	B	C	D	E
Number of organizations	45	45	45	45	60
Organizations (L,M,S)	15,15,15	29,8,8	8,29,8	8,8,29	40,10,10

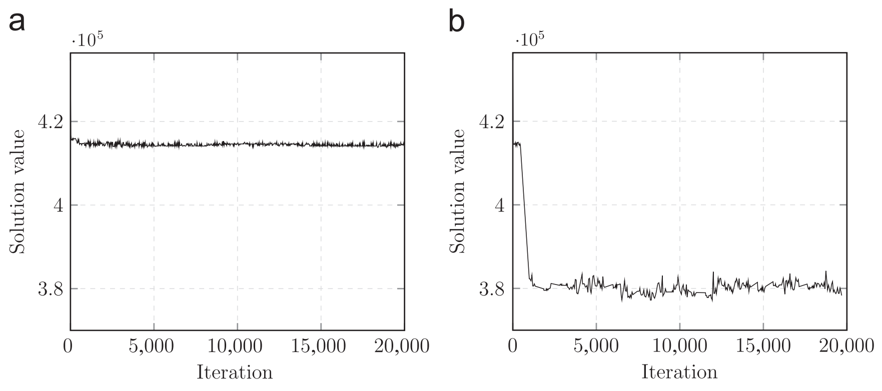

Fig. 2. Results of TS on instance 1, case A, with different values of θ. (a) $\theta=0.0005$. (b) $\theta=0.5$.

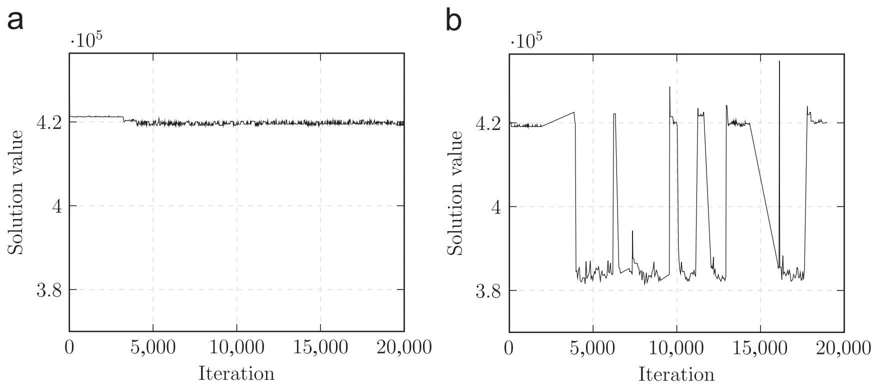

Fig. 3. Results of TS on instance 2, case A. (a) Without re-starting strategy. (b) With re-starting strategy.

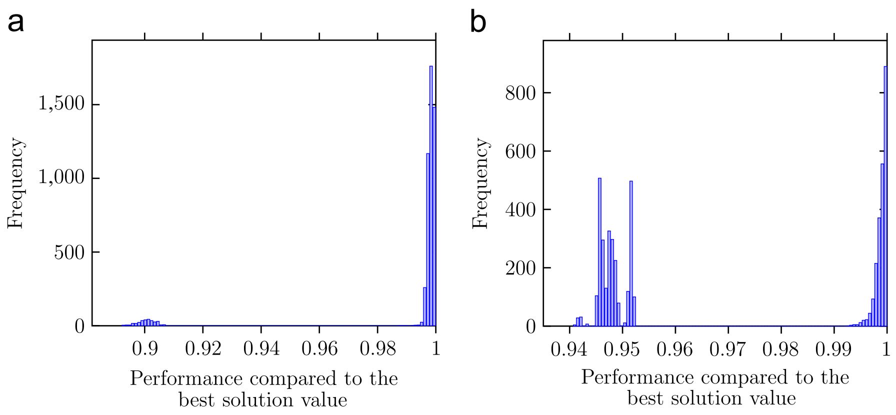

Fig. 4. Relative errors with respect to the best solutions found in 5000 runs. (a) Case A. (b) Case B.

Table 2
Average IP results after 4 h with cases A–E.

Case	A	B	C	D	E
Gap (%)	4.38	16.6	4.33	6.74	17.6
Number of iterations	19,525,040	16,596,366	25,957,733	27,008,008	9,901,677
Number of BB nodes	206,975	91,662	150,019	109,994	31,512

are quite small for cases A, C, and D (4.38%, 4.33%, and 6.74% respectively). Case B resulted in the largest average optimality gap (16.6%). This is due to the fact that there are many more large organizations in case B, which makes the instances more difficult to solve for CPLEX.

We now compare the results of the TS algorithm with the best solution obtained within 4 h by the IP model with symmetry-breaking constraints, and a simple greedy heuristic. The basic idea of the greedy heuristic is to allocate the organizations with overlapping applications on the same cluster in order to reduce the installation costs. The heuristic starts with an empty allocation. It allocates the organization that has the highest demand on an empty cluster j with the maximum capacity. Then, for cluster j and each unallocated organization k, the heuristic calculates the total installation cost of k and the organizations that have been already installed on j. Among the organizations which can fit on cluster j, the one resulting with the lowest additional installation cost on j is allocated to j. When no organizations can be assigned anymore to j without violating the capacity constraint, a new cluster with the highest capacity is opened. To fill this cluster, the allocation procedure described above is repeated. The procedure terminates when all organizations have been assigned and removes all unused capacity. A simple local search heuristic is then applied in an attempt to improve the quality of solution. This heuristic performs interchange or shift moves (as described in Section 4) until the solution does not improve anymore or a maximum of 1000 moves are performed. The performance of the greedy heuristic and of the greedy followed by the local search procedure will be discussed separately.

For the TS algorithm, we set the number of iterations to 100,000 and re-started the algorithm after 2000 iterations of non-improving solutions. The value of θ used for determining the penalty value of infeasible solutions was set to 0.5.

Table 3 contains the results of these experiments. The first two columns show the average solution values and the optimality gap obtained by CPLEX for the 10 instances of 5 different cases. The columns “Improvement (%)” present the average improvement (in percentages) on the best IP value obtained by CPLEX by the best solution value of TS, greedy without and with local search improvement. Finally, the last column reports the running times of the TS procedure (including the eventual restarts). The running times of both greedy algorithms are less than 1 s and therefore omitted from the table.

As can be seen from the table, the TS always outperformed the IP in terms of the average values of the 10 instances for all the cases (in fact, the TS obtained a better solution than the IP for each instance). From the small optimality gaps, we can conclude that the solutions obtained by both IP and TS are not far from the optimal allocations for cases A, C and D. For case B which contains many large organizations and hence is more difficult for the IP to find the optimal solutions, the improvement of TS over IP is relatively higher than the improvements that it made on cases A, C and D. In terms of computation time (in seconds per run), the running time for TS was less than 1 min, which is significantly less than the 4 h time limit of CPLEX. Additionally, we ran experiments on 10 instances of a new case E, which contains 60 organizations, and the numbers of large-, medium-, small- organizations are 40, 10, and 10, respectively. The last row of Table 3 demonstrates that the Tabu search heuristic outperformed the IP model in this case. Moreover, by comparing the results on cases E and B, the performance improvement of the TS over the IP becomes greater with the larger problem instance (1.42% vs 0.84%), within less than 1.5 min. As a local search algorithm, the performance (in terms of the solution quality and the running time) of the TS is expected to be more robust to the size of the problem, in comparison with the IP solver. The solutions obtained by CPLEX are always better than those obtained by the simple greedy heuristic alone. As cases C, D and E suggest, the greedy heuristic has higher errors when there are many small and medium organizations. The reason is that the greedy heuristic allocates organizations in the decreasing order of total demand, thus remaining at the end with many small organizations whose applications do not overlap. However, as column 6 suggests, the local search procedure leads to a better allocation of the small and medium allocations. Overall, in our experiments, both greedy algorithms produced good solutions within a very short running time (under 1 s).

Table 3
Comparison of the solutions obtained by the TS algorithm, the IP model (after 4 hours), and the greedy heuristic (without and with local search improvement) for cases A-E.

Case	Solution value (IP)	IP gap (%)	Improvement (%) (TS vs IP)	Improvement (%) (Greedy vs IP)	Improvement (%) (Greedy+LS vs IP)	Time of TS (s)
A	39,2159.6	4.38	0.57	−4.93	−0.35	45.02
B	55,3625.8	16.6	0.84	−2.14	−0.43	56.78
C	25,2250.8	4.33	0.45	−4.29	−0.73	31.41
D	22,7214.1	6.74	0.42	−5.80	−2.14	40.77
E	40,0234.8	17.6	1.42	−3.22	−0.06	84.89

5.3.1. *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 for APP. Hence, this subsection analyzes in greater detail the solutions obtained in order to gain insight into the type of organizations that are placed together on a cluster and their overlap in applications. We limit our discussion to two instances of cases B and D, where case B contains many large organizations and case D contains many small organizations.

Fig. 5a and b presents the frequencies of the applications requested by multiple organizations in the generated problem instances of case B and case D, respectively. The bar charts show the number of common applications that are requested by the organizations. Recall that there are 45 unique applications and 45 organizations in both cases B and D. For an instance of case B, each unique application is jointly requested by at least 11 organizations. There are 5 applications that are required by a large number of organizations (26–30). For most of the applications, the number of overlapped organizations is between 16 and 25. Compared to case

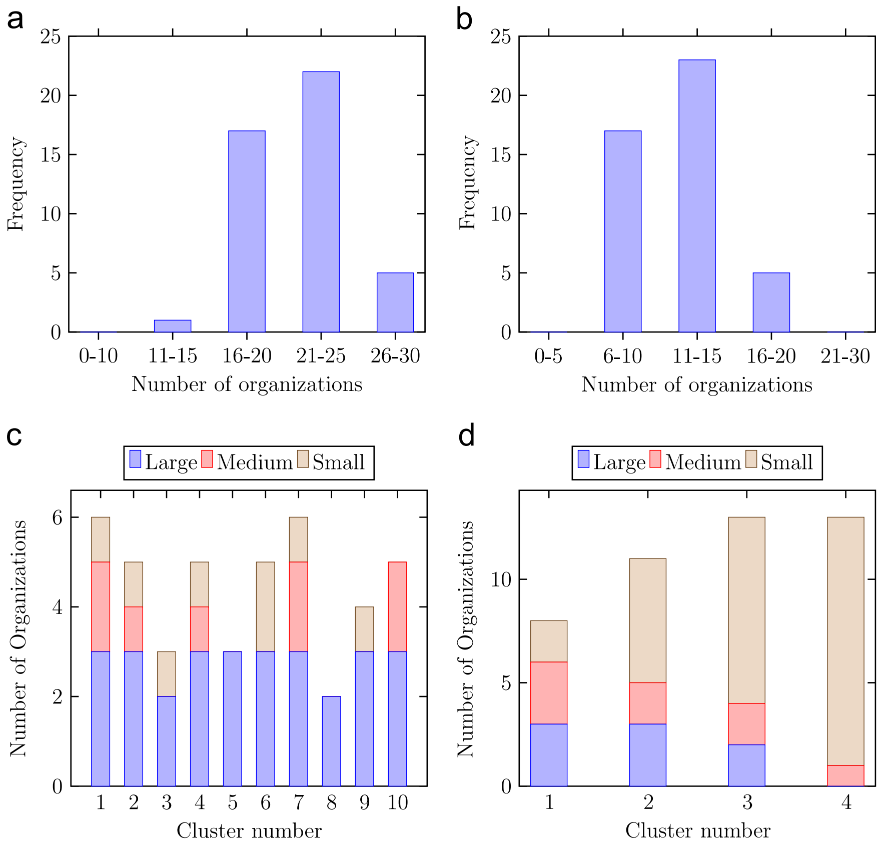

Fig. 5. Structure of the problem instance and the best solution found by TS for instance 1 of cases B and D. (a) Frequencies of the applications with overlapped organizations (Instance 1, case B). (b) Frequencies of the applications with overlapped organizations (Instance 1, case D). (c) Number of organizations per cluster (Instance 1, case B). (d) Number of organizations per cluster (Instance 1, case D).

B, in case D, there is less of an overlap of common applications among organizations. No application is requested by more than 20 organizations, and 40 applications are commonly required by between 6 and 15 organizations. The different levels of overlap present in the instances hint at the difficulty of finding good solutions for different instances. This explains why the instances of case B are the most difficult case to solve in the experiments.

Fig. 5c and d shows that in the obtained solutions, the total number of organizations per cluster and their types. In case B, the best solution opens 10 clusters, while in case D, 4 clusters were used. For both instances, the maximum capacity per cluster is installed due to the high cost of opening an extra cluster. As a consequence, the number of opened clusters is minimized and organizations are combined on a cluster for maximum overlap benefits. For both instances, the number of opened clusters depends on the number of large organizations. This is due to the fact that since capacity is limited, it is not possible to combine many large organizations on a cluster. Thus, case B, which contains 29 large applications, requires more clusters than case D. Furthermore, it seems that the algorithm fills the remaining capacity on a cluster with small and medium organizations.

5.4. Summary of the results

This subsection summarizes the results of our findings with respect to the proposed IP and Tabu Search algorithm.

- The results show that by adding symmetry-breaking constraints, a reasonable size of APP instances with 45 organizations can be solved to close to optimality with the help of an IP solver.
- The proposed penalty function and the random re-start strategy in the Tabu Search algorithm have proven their usefulness for solving a hard problem like APP. The proposed TS improved within 1.5 min the solutions given by IP at the end of 4 h.
- The proposed greedy heuristic with the local search procedure returns good solutions. The advantages of the greedy heuristic are its short computation time and its simplicity (i.e., compared to the TS, no need to set parameters). Therefore, the greedy heuristic provides a good alternative to the TS when short computation time is required.
- The results show that the difficulty of finding an optimal solution for IP depends on (1) the problem instance, and (2) the size of the problem. More specifically, the higher the number of large organizations in a problem instance and the larger the problem instance, the more difficulty for the IP to solve, and hence the lower the solution quality for cases B and E. In comparison, the performance of the TS is more robust with respect to the problem instance and the size of the problem.

  Based on these results, the IP is not recommended to compute the optimal placement for large problem instances, especially when the number of large organizations is high. The TS heuristic is able to return better solutions within a much shorter running time in all instances that we tested.

6. Conclusion

This paper has addressed the problem of placing applications on computer clusters at minimal cost. This problem is highly relevant in the design of a Software as a Service network. We proved that the problem is NP-hard, by establishing relationships with the bin-packing and graph-partitioning problems. Subsequently, we proposed a Tabu Search algorithm for finding good 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 experimental results show that the proposed Tabu Search heuristic is able to return good solutions within a short running time.

We developed heuristics for solving APP in computer clusters. A related question is whether approximation algorithms exist that can guarantee a performance bound like some special cases in the application placement problem [14]. This is not trivial, since as we discussed in Section 1, the APP has characteristics of several NP-hard problems.

Another interesting future work might involve investigating APP in game theoretical settings. From the application provider's point of view, for instance, one could be interested in how the total cost should be shared among organizations such that it is fair and in the core.

References

[1] I. Group. Software as a service, Cambridge, MA: Director Publications Ltd.; 2002.
[2] Antonopoulos N, Gillam L. Cloud computing: principles, systems and applications. London: Springer Verlag; 2010.
[3] Lomet DB. Guest editor's introduction: cloud data management. IEEE Trans Knowl Data Eng 2011;23(9):1281 URL ⟨http://dx.doi.org/10.1109/TKDE.2011.156⟩.
[4] De Jong W. The impact of a multi-cluster environment on operational costs [Master's thesis]. Erasmus University Rotterdam; 2009.
[5] Lee C. An optimal algorithm for the multiproduct capacitated facility location problem with a choice of facility type. Comput Oper Res 1991;18:167–82.
[6] Lee C. A cross decomposition algorithm for a multiproduct-multitype facility location problem. Comput Oper Res 1993;20:527–40.
[7] Mazzola J, Neebe AW. Lagrangian-relaxation-based solution procedures for a multiproduct capacitated facility location problem. Eur J Oper Res 1999;115:285–99.
[8] Melo M, Nickel S, Saldanha da Gama F. Dynammic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. Comput Oper Res 2005;33:181–208.
[9] Kang J, Park S. Algorithms for the variable sized bin packing problem. Eur J Oper Res 2003;147(2):365–72.
[10] Crainic T, Perboli G, Rei W, Tadei R. Efficient lower bounds and heuristics for the variable cost and size bin packing problem. Comput Oper Res 2011;38: 1474–82.
[11] Garey M, Johnson D. Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. San Francisco: WH Freeman and Company; 1979.
[12] Kwok T, Mohindra A. Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications. In: International conference on service oriented computing 2008; 2008. p. 633–48.
[13] Zhang Y, Wang Z, Gao B, Guo C, Sun W, Li X. An effective heuristic for on-line tenant placement problem in SaaS. In: IEEE international conference on web services 2010; 2010. p. 425–32.
[14] Urgaonkar B, Rosenberg AL, Shenoy PJ. Application placement on a cluster of servers. Int J Found Comput Sci 2007;18(5):1023–41.
[15] Sherali H, Smith J. Improving discrete model representations via symmetry considerations. Manag Sci 2001;47(10):1396–407.
[16] Jans R. Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. INFORMS J Comput 2009;21(1):123–36.
[17] Gendreau M. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003 p. 51–67.
[18] Glover F, Laguna M. Tabu search. Norwell: Kluwer Academic Publishers; 1997.
[19] Liu Z, Hacigümüş H, Moon HJ, Chi Y, Hsiung WP. PMAX: tenant placement in multitenant databases for profit maximization. In: Proceedings of the 16th international conference on extending database technology; 2013. p. 442–53.