Multi-Objective Temporal Bin Packing Problem: An Application in Cloud Computing

N. Aydin[a], I. Muter[b,*], S.I. Birbil[c]

[a] Warwick Business School, University of Warwick, Coventry, CV4 7AL, UK
[b] School of Management, University of Bath, Claverton Down, Bath BA2 7AY, UK
[c] Econometric Institute, Erasmus University Rotterdam, 3000 DR Rotterdam, The Netherlands

Abstract

Improving energy efficiency and lowering operational costs are the main challenges faced in systems with multiple servers. One prevalent objective in such systems is to minimize the number of servers required to process a given set of tasks under server capacity constraints. This objective leads to the well-known bin packing problem. In this study, we consider a generalization of this problem with a time dimension, where the tasks are to be performed with predefined start and end times. This new dimension brings about new performance considerations, one of which is the uninterrupted utilization of servers. This study is motivated by the problem of energy efficient assignment of virtual machines to physical servers in a cloud computing service. We address the virtual machine placement problem and present a binary integer programming model to develop different assignment policies. By analyzing the structural properties of the problem, we propose an efficient heuristic method based on solving smaller versions of the original problem iteratively. Moreover, we design a column generation algorithm that yields a lower bound on the objective value, which can be utilized to evaluate the performance of the heuristic algorithm. Our numerical study indicates that the proposed heuristic is capable of solving large-scale instances in a short time with small optimality gaps.

Keywords: Bin packing, cloud computing, heuristics, exact methods, column generation

1. Introduction

The assignment of a set of tasks to a set of servers permeates as an important decision in many operational problems. The utilization rate of the servers and operational expenses resulting from the

[*] Corresponding author

Email addresses: `nursen.aydin@wbs.ac.uk` (N. Aydin), `i.muter@bath.ac.uk` (I. Muter), `birbil@ese.eur.nl` (S.I. Birbil)

assignment are the main considerations in such problems. One prevalent objective in these systems is to minimize the number of servers required to process a given set of tasks under some server capacity constraints. This is also known as the bin packing problem. The servers, which correspond to bins, have limited capacity on one or more dimensions, and the tasks are characterized by the capacity that they consume on the servers. The capacity dimension of this problem forges knapsack-type constraints in the formulation, while the assignment of tasks to the servers is also required to ensure that all tasks are processed. The bin packing problem is strongly NP-Hard (Martello and Toth, 1990), and it has many applications in logistics and computer science. There is an extensive literature on the bin packing problem. For a comprehensive review of this area, we refer the reader to Delorme et al. (2016).

In this paper, we consider the temporal extension of the bin packing problem, known as the temporal bin packing (TBP) problem (Cauwer et al. (2016); Furini and Shen (2018); Dell'Amico et al. (2019)), where the tasks are to be performed in fixed start and end times during a planning period. The TBP problem is an extension of the temporal knapsack problem for which Caprara et al. (2013, 2016) proposed exact approaches based on recursive Dantzig-Wolfe decomposition (Dantzig and Wolfe (1960)). Dell'Amico et al. (2019) developed an exact algorithm for the TBP along with a number of lower and upper bounding methods. Below, we explain some of the problems that have similar characteristics to the TBP problem. In particular, we focus on cloud computing, in which the most important operational cost is the energy consumption, and propose a new extension which gives rise to an extension of TBP with multiple objectives.

Cloud computing is an internet-based computing technology which allows users to customize their on-demand computer requirements. Cloud companies provide infrastructure, platform and storage to users as accessible services in a virtual environment. A user of a cloud computing service requests a virtual machine (VM) with certain specifications (e.g. memory size, processing power and hard-disk space) for a particular time period. These virtual machines act like real computers with different operating systems and they are allocated to the physical servers in data centers. In cloud computing literature, the assignment of VM requests to physical servers is known as the virtual machine placement (VMP) problem. The scale of this problem is relatively large. In 2011, Google released its one-month cloud data which includes approximately 650k tasks assigned to 12k servers in a data center (Di et al., 2013). One of the main concerns in VMP operation is reducing the energy consumption and maximizing the effectiveness of the shared physical servers. Each data center consumes large amounts of electrical energy resulting in high operational cost and increased carbon footprint (Beloglazov, 2013). Therefore, most of the studies focus on the minimization of the number of active servers or the minimization of the operational cost of servers for the given set of VM requests (Jiang et al., 2012; Dong et al., 2013). There are several variations of this problem and, in general, the VM assignment decision does not take into consideration the temporal aspect of the customer requests. Wu et al. (2012) and Liu et al. (2018) study the energy consumption in VMP problem by assuming that all VM requests have the same arrival and departure times. The authors propose metaheuristic algorithms to tackle this problem. In a similar setting, Gao et al. (2013) investigate both energy consumption and resource wastage problems in cloud computing. They propose a multi-objective heuristic algorithm to find non-dominated solutions. Wang et al. (2019) extend the previous studies by focusing on both resource and traffic management constraints. The authors present mathematical programming based algorithms to obtain upper and lower bounds for the problem. Different than the aforementioned studies, Chaisiri et al. (2009) and Cohen et al. (2016) focus on the uncertainty in VM capacity requests. Chaisiri et al. (2009) formulate a two-stage stochastic integer programming model. While provisional capacity requests arrive in stage one, actual VM requests are allocated to servers in stage two. Cohen et al. (2016) propose a chance constrained bin packing formulation to handle uncertain demand and present several heuristics based on the proposed model.

In the literature, few studies reflect the time dimensions in their models, and tailor heuristics for the solution of this problem. Speitkamp and Bichler (2010) formulate a bin packing problem with time dimension and propose an LP relaxation-based heuristic to solve the model. The fractional assignments obtained from LP-relaxation are processed in an integer program to obtain integral assignments. The authors discuss the difficulties for solving large-size problems. Calcavecchia et al. (2012) extend the work of Speitkamp and Bichler (2010) and consider several aspects in the objective function such as demand satisfaction and load balancing among servers. They also allow to reallocate VM requests after the initial assignment. As a solution method, they propose a two-phase heuristic. While each arriving VM request is assigned to a server in the first phase, the system is periodically reviewed and reoptimized in the second phase. Cauwer et al. (2016) generalize the bin packing problem by minimizing the unused resources over time. Although the authors consider the time dimension, they assume that all VM requests arrive at the same time but can depart at different times. They provide a constraint programming implementation and discuss the difficulty of solving this problem. We refer the reader to Pires and Barn (2015) for a comprehensive review of VMP and its variations.

Time dimension in assignment of tasks to servers also arises in other areas. Vehicle scheduling problem concerns the assignment of vehicles to a set of trips that have predetermined departure and arrival locations as well as fixed start and end times. The most common cost component is associated with the travel of the vehicles between the arrival location of a trip to the departure location of the following trip without serving passengers (also known as deadheading). Analogous to the bin packing problem, the minimization of the number of active vehicles can be defined as one of the objectives (Ferland and Michelon, 1988; Dell'Amico et al., 1993). Another assignment problem involving a temporal dimension is the gate assignment problem (Mangoubi and Mathaisel, 1985). Here, the objective is to assign each aircraft departing or arriving at an airport to an available gate while maximizing both the convenience to the passengers and the operational efficiency of the airport. Essentially, these two problems are similar to the VMP when the capacity of the servers and the sizes of the tasks are equal to one. However, building a feasible assignment for these problems is fundamentally easier than that for VMP due to the capacity restriction that a single task can be assigned to a server at any time.

An extension of the vehicle scheduling problem that has both time and capacity aspects is the pick-up and delivery vehicle routing problem with time windows (Dumas et al., 1991). In this problem, each customer request is associated with an origin location, where a certain demand must be picked up, and with a destination where this demand must be delivered. The service time of these requests at the pick-up or delivery locations must start within a time-window, and the number of possible routes predominantly hinges upon the width of these windows. The case, in which the length of the time windows is zero and the routing cost is replaced by the number of vehicles in the objective, corresponds to the VMP problem. We employ this analogy later in forging our lower bounding method.

The time dimension in the assignment brings about performance considerations, one of which is the uninterrupted utilization of servers. This consideration is materialized in the vehicle scheduling problem through incorporating in the objective function the minimization of the idle time of vehicles, which is the time a vehicle is waiting at a location other than the depot. Even though these idle periods do not affect the fuel cost, this non-value added time has repercussions in terms of crew costs and inconvenience such as parking of the vehicle. The uninterrupted utilization of servers is of paramount importance in VMP problems. As for the vehicle scheduling, no direct operational cost is incurred when these servers are left idle since idle servers can be switched off to save energy. Although switching idle servers off brings energy savings, it may affect the quality of service due to the latency in reactivation (Beloglazov, 2013; Gu et al., 2018). Moreover, switch on, or fire-up, of an idle server upon arrival of a VM request consumes considerable energy (Xie et al., 2013; Fan et al., 2017). Hence, switching servers on frequently is not preferred by cloud service providers due to both energy consumption and service quality degradation. Therefore, contiguity of server utilization is desired in cloud computing, which can be achieved by minimizing the number of fire-ups alongside the number of servers utilized. In more general terms, this objective is tantamount to the minimization of the number of idle periods. Note that the durations of these idle periods, which are considered in vehicle scheduling, are not taken into account in this type of objective.

In this paper, we model and tackle the VMP problem, which is intrinsically similar to the temporal bin packing problem. We make the following contributions: To capture both aspects of utilization discussed above, we define two objectives, namely minimizing the number of utilized servers and minimizing the number of fire-ups. These objectives are inherently correlated as the minimization of the number of active servers may result in enhanced utilization of the center (less idle periods). However, we show that minimizing the number of active servers does not necessarily minimize the number of fire-ups or vice versa. We employ a prominent technique in multi-objective optimization, namely the weighted sum method, that leads to a mixed-integer programming model with a single objective. By analyzing the structural properties of the problem, we propose an efficient heuristic method for large-scale problems. This heuristic is based on eliminating fire-ups by solving smaller optimization models. In order to evaluate the performance of this heuristic, we develop an exact lower bounding method based on column generation, which is also used to obtain an upper bound. The novelty of this method lies in the way the temporal aspect is incorporated into the pricing subproblem. With our numerical study, we demonstrate that the proposed methods achieve strong bounds for medium- to large-scale VMP instances.

2. Problem Formulation

In this section, we present a mathematical model for the VMP problem. We focus on two objectives that have been considered to optimize the efficiency of the hosting systems, namely the minimization of the number of servers and the minimization of the number of fire-ups. We analyze the characteristics of these objectives and discuss the relationship between them. Let $I = \{1, 2, .., n\}$ be the set of VM requests to be hosted on a set of m identical physical servers indexed by K. The planning horizon is of length T, and each VM request $i \in I$ requires a certain amount of capacity, $c_i \in \mathbb{Z}^+$. Moreover, each VM $i \in I$ resides in the system for the duration within the requested time interval $[s_i, e_i]$, where $s_i \in \mathbb{Z}^+ \cup \{0\}$ and $e_i \in \mathbb{Z}^+$ are the start and the end times of request $i \in I$, respectively, and $e_i > s_i \geq 0$ holds. An arriving request is assigned to one of the servers $k \in K$, which has an available capacity to host this request. Although, in cloud computing, VM requests have various capacity requirements, such as CPU and memory, one resource is generally binding with respect to others (Cohen et al., 2016). Hence, we assume the servers are limited by a single resource, and the capacity of each server is denoted by C, which is also assumed to be at least one, i.e. $C \geq 1$. An idle server is switched on when an arriving VM request is assigned to it. Otherwise, the server stays in stand-by mode. We make server allocation decisions by considering the requested time intervals for the VM requests. We define binary variables x_{ik} to denote the VM assignment decisions, where x_{ik} is equal to one only if VM $i \in I$ is assigned to physical server $k \in K$. The assignment of a VM to a server consumes the capacity of the server from the start to the end times of this request. Thus, the capacity consumption of a server is affected only when a new VM enters the system or an existing one departs from it. We define the index set τ to mark the VM start and end times. That is, $l \in \tau$ assists to denote the start or the end time of a VM request such that we obtain an ordered set of times, $t_l > t_{l-1}$ for $l > 1$. To designate the existence of a VM, we use a binary parameter a_{il} that is set to one only if VM request i exists at time $t_l \in \{0, \cdots, T\}$, $l \in \tau$. We assume that each VM request arrives and leaves at the beginning of a time period. Therefore, we have $a_{il} = 0$ for $l = e_i$. To determine whether a physical server is used at all during the planning horizon, we define the binary decision variable z_k, which takes value one only if physical server $k \in K$ is used. The mathematical programming model then becomes

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in K} z_k && && (1)\\
\text{subject to} \quad & \sum_{i \in I} a_{il} c_i x_{ik} \leq z_k C, && k \in K; l \in \tau_A, && (2)\\
& \sum_{k \in K} x_{ik} = 1, && i \in I, && (3)\\
& x_{ik}, z_k \in \{0, 1\}, && i \in I; k \in K, && (4)
\end{aligned}$$

where $\tau_A \subseteq \tau$ denotes the index set of VM start times. Constraint set (2) ensures that the total load on a physical server does not exceed its capacity. Constraints (3) guarantee that each VM request is assigned to a physical server. This model is tantamount to the temporal bin packing problem where the capacity restriction is imposed for a set of time periods corresponding to the arrival times of VM requests.

Another important determinant of energy consumption is the number of times the machines are switched on from the stand-by mode, in which the servers are at the maximum energy savings mode. It has been pointed out by Fan et al. (2017) that the larger the number of fire-ups, the higher the energy consumption is. To find out which servers are in use at time t_l, we first define a binary decision variable y_l^k taking value one if and only if the physical server k is on at time t_l. Then, we identify the fire-up of a server by checking its condition at two consecutive time periods. We define w_l^k to denote the number of fire-ups on server k at time t_l for $l \in \tau_A$. We note that a server can be switched on when a new request arrives so that w_l^k is defined only for arrival times. If the difference $y_l^k - y_{l-1}^k$ for $l > 1$ is equal to one, then this means that we start server k at time t_l and $w_l^k = 1$. Consequently, the mathematical model that minimizes the number of fire-ups of the physical servers is given by

$$\begin{aligned}
\text{minimize} \quad & \sum_{l \in \tau_A} \sum_{k \in K} w_l^k && && (5)\\
\text{subject to} \quad & y_l^k \leq \sum_{i \in I} a_{il} c_i x_{ik} \leq y_l^k C, && k \in K; l \in \tau, && (6)\\
& \sum_{k \in K} x_{ik} = 1, && i \in I, && (7)\\
& y_l^k - y_{l-1}^k \leq w_l^k, && k \in K; l \in \tau_A : t_l = s_i, i \in I, && (8)\\
& x_{ik}, y_l^k \in \{0, 1\}, && i \in I; k \in K; l \in \tau, && (9)\\
& w_l^k \geq 0 && k \in K; l \in \tau_A. && (10)
\end{aligned}$$

Unlike (1) - (4), this model features $w-$variables and the set of constraints (8), and the capacity constraints (6) are imposed for both the arrival and departure times of the VM requests. Constraint set (6) also ensures that the physical server is marked as idle (switched-off) if it is not in use at time $t_l, l \in \tau$. In order to formulate constraints (8), we define a dummy variable y_0^k and set it to zero. Note that there is only non-negativity constraints on w_l^k since it can only be zero or one so long as y variables are binary. This helps us reduce the number of binary variables.

Although two objectives, namely the minimization of the number of active servers and the minimization of the number of fire-ups, are intrinsically similar, considering only one of them does not necessarily yield the optimum for the other. The counterexamples below attest to the need for an objective function incorporating both criteria.

Example 2.1. *Figure 1 illustrates the first counterexample, where the optimal solution of model (1) - (4) is not optimal for model (5) - (10). The servers in the figure are identical and each has the capacity of three units. The optimal assignment is computed for three VM requests with the* *objective of minimizing the number of servers. The problem data are given in the left-hand side of Figure 1. At the optimal solution, two servers are required to schedule the three VM requests. In*

VM No	Capacity Request
I	2
II	2
III	1

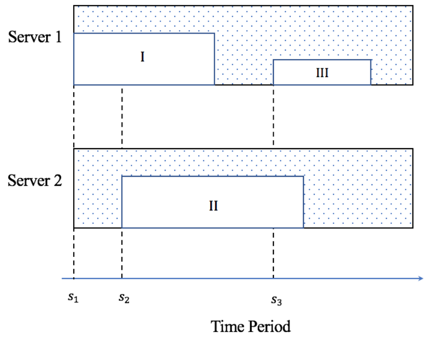


Figure 1: An example demonstrating that the optimal solution of (1) - (4) is not optimal for (5) - (10)

this schedule, the total number of start-ups is three. Although this solution is optimal for model (1) - (4), it is not optimal for model (5) - (10). We can improve the number of start-ups by assigning VM request III to server 2. In the resulting schedule, the total number of start-ups becomes two.

Figure 2 illustrates the second counterexample, where the optimal solution of model (5) - (10) is not optimal for model (1) - (4). Here, two VM requests arriving in separate time intervals are placed in two servers. In the optimal solution, the total number of start-ups is two. However, this schedule requires two physical servers. We can improve the number of required physical servers by moving VM request II to server 1. In the resulting schedule, the total number of servers becomes one.

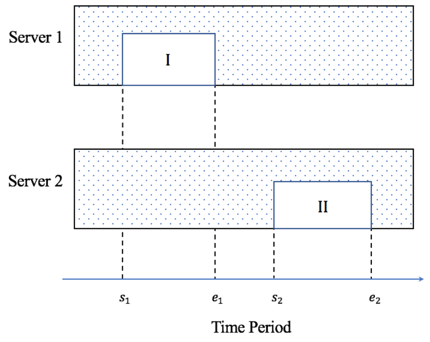


Figure 2: An example demonstrating that the optimal solution of (5) - (10) is not optimal for (1) - (4)

Thus, the minimization of the number of fire-ups and the number of active servers should be considered simultaneously in the objective function to capture the needs of the hosting service. Mingling these two objectives using weighted sum method, we obtain the following optimisation model:

$$
\begin{aligned}
(\mathbf{M1})\quad \text{minimize}\quad & \gamma \sum_{l\in\tau_A}\sum_{k\in K} w_l^k + \sum_{k\in K} z_k && && (11)\\
\text{subject to}\quad & y_l^k \le \sum_{i\in I} a_{il} c_i x_{ik} \le y_l^k C, && k\in K; l\in\tau, && (12)\\
& \sum_{k\in K} x_{ik} = 1, && i\in I, && (13)\\
& y_l^k - y_{l-1}^k \le w_l^k, && k\in K; l\in\tau : t_l = s_i, i\in I, && (14)\\
& y_l^k \le z_k, && k\in K; l\in\tau, && (15)\\
& y_l^k \ge x_{ik}, && i\in I; k\in K, l\in\tau : t_l = s_i, && (16)\\
& x_{ik}\in\{0,1\}, && i\in I; k\in K, && (17)\\
& y_l^k\in\{0,1\}, w_l^k \ge 0, && k\in K; l\in\tau, && (18)
\end{aligned}
$$

where $\gamma > 0$ is a scaling parameter. Constraints (12)-(14) are similar to constraints (6)-(8). Constraints (15) are defined to keep track of the number of physical servers scheduled throughout the planning horizon. Constraints (16) are redundant due to (12), however, they tighten the LP relaxation of the model. Bin packing problems are usually highly symmetric and LP relaxation of the problem provides a weak lower bound. The optimal server allocations obtained by model M1 can be used to build equivalent solutions by just permuting the servers. In order to reduce the symmetry of the solution space and tighten the LP relaxation bound, we introduce additional constraints as the ones in (16) to the model M1. Given the requested capacities for the virtual machines, we can compute a lower bound on the number of required physical servers as follows,

$$h = \left\lceil \max_{l\in\tau}\{\sum_{i\in I} a_{il} c_i / C\} \right\rceil. \qquad (19)$$

This lower bound can be used to select some of the physical servers for VM assignment. We enforce the first h servers to be used for the allocation by introducing the constraint $z_1 + \cdots + z_h = h$. In addition, we can also fix the order of the servers used for the assignment with the constraints $z_{k+1} \le z_k, k \in K$. These constraints ensure that server $k+1$ can be only used if the server k is allocated before.

Next, we analyze the relationship between two objective functions in model M1. Under certain conditions, minimizing the number of fire-ups together with the number of scheduled servers does not change the minimum number of servers that can be obtained by solving the single objective model (1) - (4).

PROPOSITION 2.1. *For any $\gamma > 0$, the optimal solution of* (11) - (18) *is also optimal for* (1) - (4) *when $\sum_{k\in K} z_k \leq 2$.*

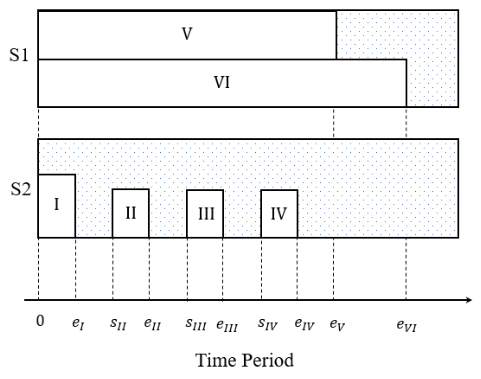
Proof. Let Z^* and W^* be the optimal number of servers and optimal number of fire-ups obtained by solving the problem (11) - (18) for $\gamma = 1$, respectively. We first show for $\gamma = 1$ that there is no feasible solution with one server, when the optimal number of servers Z^* is greater than one. This shall follow from contradiction. Suppose that $Z^* = 2$ and there are total W^* fire-ups on these two servers. Assume that there is a feasible solution with one server and $\hat{W}$ fire-ups. When we move the assigned VM requests on server 2 to server 1, some requests may overlap and hence, the resulting number of fire-ups in the feasible solution would be less than W^*. Consequently, the objective value of the feasible solution, $1 + \hat{W}$, is less than the optimal objective value, $2 + W^*$, and this contradicts the optimality of $Z^* = 2$ and W^*.

The number of fire-ups increases, if we schedule the same number of VM requests to two physical servers instead of one. Therefore, the optimal number of servers obtained by solving (11) - (18) is the same as that obtained by solving (1) - (4) for any $\gamma > 0$ when the optimal number of servers are less than or equal to two for model (11) - (18). □

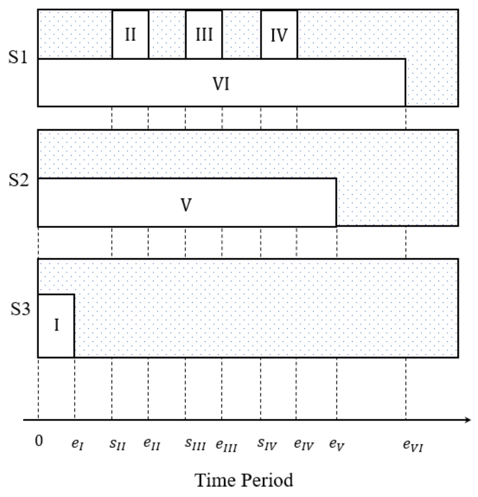
Unfortunately, this result cannot be generalized. The counterexample below shows that number of fire-ups can be further decreased when we increase the number of servers.

EXAMPLE 2.2. *Figure 3 illustrates the counterexample, where the minimum number of active servers obtained by* (11) - (18) *is not optimal for model (1) - (4). For this example, we set $\gamma = 1$. The servers denoted by $S1, S2$ and $S3$ in the figure are identical and each has the capacity of four units. During the planning horizon six VM requests arrive. While the capacity requirement for VM I is three units, the remaining VMs require two units. The optimal assignment for model (1) - (4) is given in Figure 3(a), where the optimal number of servers is two and the resulting number of fire-ups is five. On the other hand, the optimal assignment for model* (11) - (18) *shown in Figure 3(b) has three fire-ups and uses three servers. In order to decrease total number of fire-ups and active servers, model* (11) - (18) *allocates six VM requests to three servers instead of two.*

Since a virtual machine request may lead to a fire-up, the total number of fire-ups is at most the number of virtual machine requests which is denoted by n. In order to ensure that the optimal



(a) Optimal assignment for model (1) - (4)



(b) Optimal assignment for model (11)-(18)

Figure 3: An example demonstrating that the optimal number of servers obtained by (11)-(18) is not optimal for (1) - (4)

number of active servers obtained by model (M1) is equal to the optimal number of servers obtained by single objective model (1) - (4), we set $\gamma = 1/n$. This is tantamount to the lexicographic approach in multi-objective optimisation, which minimises objectives in a predefined order. Therefore, the minimum number of fire-ups is obtained for the minimum number of machines. In Section 5, we also solve M1 by setting $\gamma = 1$ so as to evaluate the impact of this coefficient on the number of active servers and fire-ups.

3. Bounding Methods

Being an extension of the VMP problem, M1 is also NP-hard, which makes it important to find efficient solution strategies for large-scale problems. In this section, by analyzing the structural properties of the problem, we propose methods that provide upper bounds on the VMP problem.

Heuristic algorithms, such as best fit and first fit, are considered to be efficient solution strategies for the VMP problem in the literature (Cohen et al., 2016; Speitkamp and Bichler, 2010). These methods iteratively allocate the arriving VM requests according to the size of the remaining server capacities. As an initialization step, we apply a new constructive heuristic based on the best fit procedure. To improve the initial solution, we propose an heuristic based on the iterated neighbourhood search.

3.1. Constructive Look-ahead Heuristic (CLH)

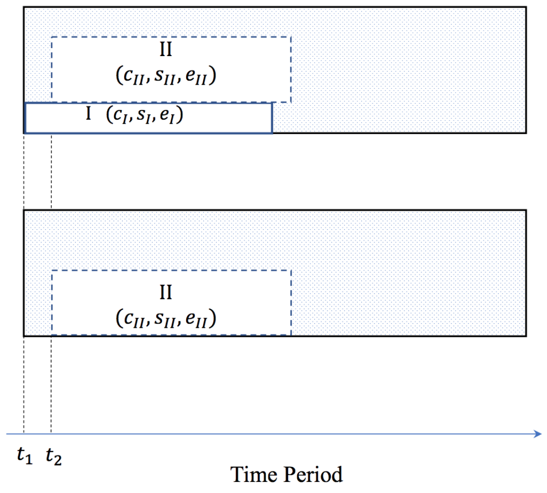
Assigning VM requests to physical servers can be considered as a sequential decision-making problem where VM requests are assigned one at a time. To schedule the VM requests, we order them according to their arrival time such that $s_1 \leq s_2 \leq \cdots \leq s_m$. At each time t_l for $l \in \tau_A$, we assign a VM request to a physical server. Given the state of the system at time t_l, look-ahead placement heuristic allocates the arriving VM request to a server by considering the future VM requests arriving at the next k time periods such that $t_{l+1}, ..., t_{l+k}$. This concept is illustrated in Figure 4 for $k = 2$ for simple representation. In our numerical experiments, we set $k = 3$. VM request I has been already placed in the first server. At time period t_2, VM request II arrives to the system. By considering the remaining capacity of working server(s), we determine all feasible positions for the arriving request. As depicted in Figure 4(a), there are two feasible positions to place VM request II. We can assign it to either currently active server 1 or to a new server 2. By considering the VM requests III and IV arriving at time periods t_3 and t_4, respectively, we decide the best position which minimizes the objective function in model M1. Note that in order to compute the objective function value for each feasible position of VM request II, we temporarily assign VM requests arriving at time periods t_3 and t_4 based on best fit heuristic. As illustrated in Figure 4(b), placement of VM request II to server 1 results in three fire-ups and requires three physical servers. On the other hand, by scheduling it to server 2, we can use two physical servers which result in two fire-ups. For this example, we place VM II to server 2. When we are indifferent between possible placement options, we use the generic best fit approach and assign the VM request to the server with the lowest remaining capacity.

3.2. Recovery Algorithm

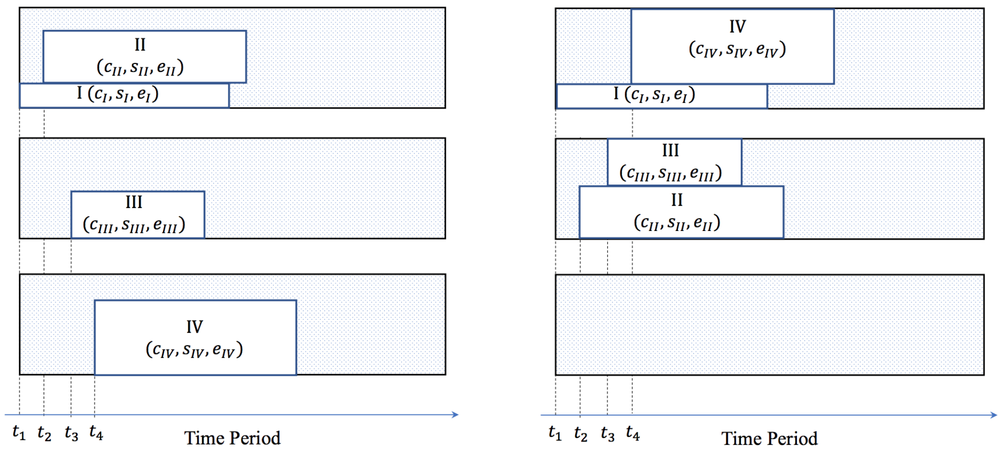
To improve on the solution from CLH, we design a simple and efficient heuristic which is based on the combination of iterated neighbourhood search and mathematical programming. The solution obtained from CLH is used as the initial solution. At each iteration, we define a neighbourhood and reschedule the VM requests only in this neighbourhood by solving small- to medium-scale M1 problems.

Algorithm 1 gives the steps of the proposed recovery heuristic. The main idea behind this approach is to eliminate the unnecessary fire-ups by rescheduling the VM requests initiating a fire-up in the current solution. Thus, at each iteration, we find the eligible VM requests that cause a fire-up to construct a neighbourhood (lines 3-5 and 10-14). One of the eligible VM requests is selected and

VM No	Properties
II	(c_{II}, s_{II}, e_{II})
III	$(c_{III}, s_{III}, e_{III})$
IV	(c_{IV}, s_{IV}, e_{IV})



(a) Initial System


II
(c_{II}, s_{II}, e_{II})
I (c_I, s_I, e_I)
III
$(c_{III}, s_{III}, e_{III})$
IV
(c_{IV}, s_{IV}, e_{IV})
t_1 t_2 t_3 t_4
Time Period
IV
(c_{IV}, s_{IV}, e_{IV})
I (c_I, s_I, e_I)
III
$(c_{III}, s_{III}, e_{III})$
II
(c_{II}, s_{II}, e_{II})
t_1 t_2 t_3 t_4
Time Period

(b) Assignment Options

Figure 4: An illustration of VM assignment process in CLH

other VM requests overlapping with the selected VM (first degree neighbours) or arriving/leaving within a predefined time interval (second degree neighbours) are identified. By selecting some of these VM requests randomly, a neighbourhood is constructed (lines 16-18). Note that by including the second degree neighbours, we increase the variability in the constructed neighbourhood which decreases the risk of searching the solution in the same area.

The VM requests in the generated neighbourhood are removed from their assigned physical servers in the current solution and rescheduled by solving a restricted version of model M1, where VMs outside the neighbourhood are set to their current assignment. The current solution is updated with the new schedule and the process is repeated until a stopping criterion is met. The proposed algorithm is terminated if the solution is not improved in a specified number of iterations. The overall structure of the algorithm is displayed in Algorithm 1.

The key component of this method is to generate a neighbourhood which helps to improve the solution quickly. To increase the efficiency of the proposed algorithm, unnecessary calls to the subproblem (model M1) should be eliminated, such as duplicated examination and rescheduling of the VM requests in an unpromising neighbourhood. Therefore, we use tabu list strategy as in the tabu search procedure (Glover, 1989). Recall that a neighbourhood is generated based on the selected VM. In order to avoid rescheduling in the same neighbourhood, the selected VM is kept in the tabu list for a certain number of iterations (line 13). In addition, to boost the speed of our algorithm in the first iteration, we select the VM which has the most fire-ups in its adjacency set (lines 6-7). This step helps us to quickly eliminate some of the unavoidable fire-ups.

4. Column Generation

In this section, we present a column generation algorithm to find a tight lower bound and an upper bound for the VMP problem. To facilitate the demonstration and pave our way to the presentation of this algorithm, we provide an alternate formulation for the VMP problem. We note that the start of a request i on server k triggers a fire-up if i is the only VM on server k at its start time s_i. This observation obviates the need for $y-$variables, which are used along with the set of constraints (12) to indicate whether a server is used. In order to translate the load on a server into a constraint *without* using the time index $l \in \tau$, we first put the requests in ascending order of their start times; that is, $i < j$ if $s_i < s_j$ and ties are broken arbitrarily. We define two sets $\delta_i = \{j | j < i : e_j > s_i\}$ and $\delta_i^+ = \{j | j < i : e_j \geq s_i\}$. While the former set contains the requests that are active at the start time of i, the latter is similar except that it includes those requests whose

Algorithm 1: Recovery Algorithm

```
1: Obtain initial solution $(\hat{x}, \hat{y}, \hat{w}, \hat{z})$ from CLH
2: for $i \in I$ do
3:     Construct adjacency set $\mathcal{A}_i = \{j \in I : s_i \leq s_j \leq e_i \vee s_i \leq e_j \leq e_i\}$
4: Initialize tabu list
5: Initialize $numiter = 1$
6: while stopping criterion not met do
7:     For each $i \in I$, define $w_i^k = \hat{w}_l^k$ such that $l = s_i$
8:     Define fire-up set $\mathcal{S} = \{i \in I : w_i^k = 1, k \in K\}$
9:     if $numiter = 1$ then
10:        Select VM $j$ with the highest number of fire-ups in its adjacency set such that
           $j = \arg\max_{j \in \mathcal{S}} \{\sum_{k \in K} \sum_{i \in \mathcal{A}_j} w_i^k\}$
11:    else
12:        Randomly select a VM $j$ which is not tabu from set $\mathcal{S}$
13:    Add $j$ to tabu list; update tabu list
14:    Randomly select a subset of first degree neighbours $\mathcal{R}_1 \subset \mathcal{A}_j$
15:    Find the time interval $(s, e)$ for set $\mathcal{R}_1$
16:    Randomly select a subset of second degree neighbours
        $\mathcal{R}_2 \subset \{i \in I \setminus \mathcal{A}_j : s \leq s_i \leq e \vee s \leq e_i \leq e)$
17:    Set $x_{ik} = \hat{x}_{ik}$ for $i \in I \setminus (\mathcal{R}_1 \cup \mathcal{R}_2)$ in M1
18:    Solve model M1 and obtain new solution $(\hat{x}, \hat{y}, \hat{w}, \hat{z})$
19:    $numiter = numiter + 1$
20: return solution $(\hat{x}, \hat{y}, \hat{w}, \hat{z})$
```

end times are equal to the start time of i. This nuance is necessary due to the aforementioned assumption, which stipulates that request i does not trigger a fire-up on server k, if there is request j on k with $e_j = s_i$ even though j does not consume capacity at its end time. Therefore, $j \in \delta_i^+$ but $j \notin \delta_i$, when $e_j = s_i$.

Thus, the resulting model is written as

$$
\begin{aligned}
&(\mathbf{M2}) \text{ minimize} && 1/n \sum_{l \in \tau_A} \sum_{k \in K} w_l^k + \sum_{k \in K} z_k && && (20)\\
&\text{subject to} && \sum_{j \in \delta_i} c_j x_{jk} + c_i x_{ik} \le C z_k, && k \in K, i \in I, && (21)\\
& && x_{ik} \le z_k, && k \in K; i \in I, && (22)\\
& && \sum_{k \in K} x_{ik} = 1, && i \in I, && (23)\\
& && \sum_{j \in \delta_i^+} x_{jk} - x_{ik} + w_l^k \ge 0, && k \in K; i \in I; l \in \tau_A : t_l = s_i, && (24)\\
& && x_{ik}, z_k \in \{0,1\}, && i \in I; k \in K, && (25)\\
& && w_l^k \ge 0, && l \in \tau_A; k \in K, && (26)
\end{aligned}
$$

where (23) stays intact, and the other constraints and variables undergo some changes in this new formulation. First, by stripping the model of $y-$variables and time indices, the server utilization is represented solely by $z-$variables. Constraint set (21) limits the total load on server k at the start time s_i of each request i, which is the total size of the requests in δ_i plus c_i, to the server capacity. Though implied by (21), constraints (22) are added to improve the LP relaxation of MP2. Constraint set (24) imposes that variable w_l^k takes value one if at $t_l = s_i$ for a request i, which is assigned to server k, no request in δ_i^+ is assigned to this server. As alluded to previously, a request j, for which $s_i = e_j$ holds, does not consume capacity at s_i, and also, does not cause a fire-up at s_i, if i is the only active request at that time.

Note that in M2, instead of the time index set with cardinality $|\tau| = 2n$, the capacity and fire-up controls are both handled at the start time of the requests whose cardinality is n. Comparing these two models M1 and M2, we conclude that in the latter, both the number of variables and the number of constraints shrink by $3nm$. Although M2 is considerably more economical than M1 in terms of both constraints and variables, we observe that its performance is inferior due to its poor LP relaxation. In the next proposition, we show the strength of M1 over M2.

PROPOSITION 4.1. *The LP relaxation bound of M1 is at least as large as that of M2.*

Proof. We show that any feasible LP solution of M1 is also feasible for M2, but not vice versa. To prove the former, suppose that $(\hat{x}, \hat{w}, \hat{y}, \hat{z})$ is a feasible solution of the LP relaxation of M1. $\hat{x}$ and $\hat{z}$ satisfy (21) due to (12) where $\{j \in I : a_{jl} = 1\} = \delta_i \cup \{i\}$ for $t_l = s_i$ and (15). Moreover, (22) is implied by (15) and (16). Feasibility of (23) is trivial as it is equivalent to (13). To show feasibility of (24), we first lay out two equalities resulting from the LP relaxation solution of M1, which are as follows for a given $k \in K$ and $l \in \tau$:

$$\hat{y}_l^k = \max\left\{\max_{i:a_{il}=1}\{\hat{x}_{ik}\}, \frac{\sum_{i\in I} a_{il} c_i \hat{x}_{ik}}{C}\right\}, \tag{27}$$

$$\hat{w}_l^k = \max\{0, \hat{y}_l^k - \hat{y}_{l-1}^k\}, \tag{28}$$

where the former is due to (12) and (16) and the latter follows from (14). Substituting the terms, we can rewrite (24) for $i \in I$, $k \in K$ and $t_l = s_i$ as

$$w_l^k \geq \hat{y}_l^k - \hat{y}_{l-1}^k \geq \hat{x}_{ik} - \sum_{j\in\delta_i^+} \hat{x}_{jk}. \tag{29}$$

We arrange (29) as

$$\hat{y}_l^k - \hat{x}_{ik} \geq \hat{y}_{l-1}^k - \sum_{j\in\delta_i^+} \hat{x}_{jk}, \tag{30}$$

where $\sum_{j\in\delta_i^+} \hat{x}_{jk} = \sum_{j\in I} a_{jl-1}\hat{x}_{jk}$ for $t_l = s_i$, because all the requests active at t_{l-1} also reside in δ_i^+, i.e. there is no start or end event between t_{l-1} and t_l. Plugging this equality in (27), one of the following holds: $\hat{y}_{l-1}^k = \max_{j:a_{jl-1}=1}(\hat{x}_{jk})$ or $\hat{y}_{l-1}^k = \frac{\sum_{j\in I} a_{il-1} c_j \hat{x}_{jk}}{C} = \frac{\sum_{j\in\delta_i^+} c_j \hat{x}_{jk}}{C}$. Therefore, we can show that (30) is satisfied using the following two inequalities:

$$\hat{y}_l^k - \hat{x}_{ik} \geq \max_{j:a_{jl-1}=1}\{\hat{x}_{jk}\} - \sum_{j\in\delta_i^+} \hat{x}_{jk}, \tag{31}$$

$$\hat{y}_l^k - \hat{x}_{ik} \geq \frac{\sum_{j\in\delta_i^+} c_j \hat{x}_{jk}}{C} - \sum_{j\in\delta_i^+} \hat{x}_{jk}. \tag{32}$$

In both inequalities, the left-hand-side is non-negative due to (16). In (31), the right-hand-side is non-positive since $\arg\max_{j:a_{jl-1}=1}\{\hat{x}_{jk}\} \in \delta_i^+$, while that in (32) is non-positive as $c_j/C \leq 1$, $j \in \delta_i^+$.

Now, we show that there exists a feasible LP solution for M2 that is not feasible for M1. To that end, we give a simple counterexample with $m = 2$, $n = 3$, $C = 1$, $k \in K$ and $c_i = 1$, $i \in I$, for which $\hat{x}_{ik} = 1/2$ for all $i \in I$ and $k \in K$. The solution is demonstrated in Figure 5. These $x-$values induce $\hat{z}_1 = 1$ and $\hat{z}_2 = 1$ due to constraints (21)-(22), which render $\hat{z}_k = \max_i \left\{ \hat{x}_{ik}, \frac{\sum_{j \in \delta_i} c_j \hat{x}_{jk}}{C} \right\}$. From (24), for $i \in I \backslash \{1\}$ and $k \in K$, the value of w_l^k, $t_l = s_i$ can then be found by $\hat{w}_l^k = \max \left\{ 0, \hat{x}_{ik} - \sum_{j \in \delta_i^+} \hat{x}_{jk} \right\}$, which yields $\hat{w}_0^1 = \hat{w}_0^2 = 1/2$ and $\hat{w}_i^k = 0$, otherwise. The corresponding objective function value is 7/3. This solution can be translated into the solution of M1 for x, w and z variables directly, and we have to check whether they are feasible for M1 by deducing the values of y through its constraints. Due to (16), $\hat{y}_l^k \geq 1/2$ for $t_l = 0, 1, 2$ and $k = 1, 2$. Along with these constraints, (14) stipulates that $y_0^1 = y_1^1 = 1/2$. However, these values violate (12) for $k = 1$ and $t_l = 1$ because $c_1 x_{11} + c_2 x_{21} = 1 > C y_1^1 = 1/2$. In fact, the given values of $x-$variables in the solution of M1, namely $\hat{x}_{ik} = 1/2$ for all $i \in I$ and $k \in K$, induce $y_0^1 = y_0^2 = y_3^1 = y_3^2 = 1/2$, $y_1^1 = y_1^2 = y_2^1 = y_2^2 = 1$, $z_1 = z_2 = 1$ and $\hat{w}_0^1 = \hat{w}_1^1 = \hat{w}_0^2 = \hat{w}_1^2 = 1/2$, and all the other variables assume value zero. The corresponding objective function value of M1 is 8/3, which is larger than that of M2.

VM No	(s_i, e_i)
1	$(0, 2)$
2	$(1, 3)$
3	$(2, 4)$

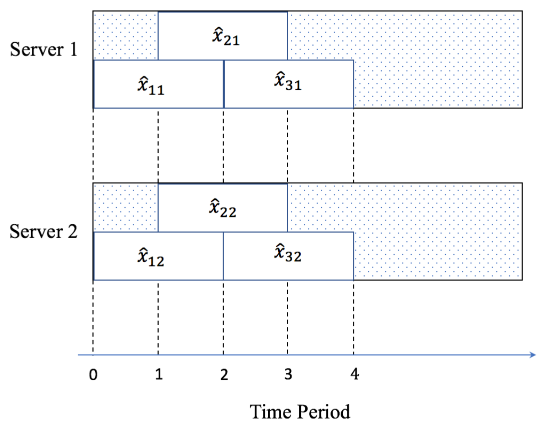


Figure 5: An example demonstrating that a feasible solution of M2 is not feasible for M1

$\square$

Constraint sets (21) and (24) exhibit a block-angular structure, which is decomposable for each server k. Each solution satisfying (21) and (24) for a server k is a feasible schedule, which satisfies the capacity constraints and those constraints facilitating the calculation of fire-ups. Applying the steps of Dantzig-Wolfe decomposition (De Carvalho, 2002), which is similar to those for the bin packing problem, and hence omitted here, we can reformulate this problem, referred to as the master problem (MP), as follows:

$$
\begin{aligned}
(\mathbf{MP}) \quad \text{minimize} \quad & \sum_{s\in S} r_s \lambda_s && && (33)\\
\text{subject to} \quad & \sum_{s\in S} \alpha_{is}\lambda_s = 1, && i \in I, && (34)\\
& \sum_{s\in S} \lambda_s \leq m, && && (35)\\
& \lambda_s \in \{0,1\}, && s \in S, && (36)
\end{aligned}
$$

where S is the set of all feasible server schedules, and binary variable λ_s is equal to one, only if schedule s is selected. For a given server schedule s, the parameter r_s is its cost, which is one plus the fire-up cost, and α_{is} is a binary parameter taking value one, only if request i is covered in schedule s. While constraint set (34), which is the counterpart of (23) after decomposition, ensures that each request exists in exactly one of the selected schedules, constraint set (35) limits the number of selected schedules to the number of available servers.

MP is a set partitioning problem with an extra constraint (35), which contains an exponential number of variables due to the size of S. Enumerating S completely would be prohibitive even for small instances. The prominent method to go about solving the LP relaxation of such problems is column generation, which initializes a restricted MP (RMP) by replacing S with its subset $\bar{S}$ and by dispensing the integrality constraint (36). The solution of RMP offers the dual information associated with the constraint sets (34) and (35), which can then be utilized in a pricing subproblem (PSP) to check dual constraints corresponding to the schedules. If the PSP discovers a violated dual constraint, in other words a schedule with a negative reduced cost, it is added to the RMP, and the same steps ensue. Otherwise, the column generation algorithm stops at the optimal LP relaxation solution of the MP. We remark that the LP relaxation of MP is stronger than that of M2 since the former results from applying Dantzig-Wolfe decomposition to the latter. In Section 5, we report the LP relaxation bound of MP obtained by column generation and that of M1, and show that, on the average, MP also leads to stronger LP bounds than M1. To reach the optimal integral solution of the MP, this column generation method must be embedded in a branch-and-bound tree, which is known as branch-and-price (Barnhart et al., 1998). In this paper, we apply the column generation algorithm without embedding it in a branch-and-bound tree. We initialize $\bar{S}$ with the schedules produced by the recovery heuristic. The output of the column generation algorithm is a lower bound, which is used to evaluate the performance of our proposed heuristic, and a set of columns along with their fractional values. We incorporate these columns to form a reduced form of MP with integrality constraints and solve it using a mixed integer programming (MIP) solver to obtain an upper bound for the VMP problem. Therefore, it attains an upper bound at least as good as the recovery heuristic.

In the rest of the section, we explain the PSP and propose an algorithm to solve it exactly. The objective of this problem is to find the schedule with the smallest reduced cost, which can be posed as

$$\min_{s \in S}\{r_s - \sum_{i \in I} u_i \alpha_{is} - v\}, \tag{37}$$

where u_i, $i \in I$ are the dual variables associated with the constraints (34), v is the dual variable associated with the constraint (35) and r_s is a function of the selected requests, which are indicated henceforth by binary variable x_i. A selected request i, which is assigned when a server is idle, induces a fire-up, rendering binary variable $w_l = 1$, $t_l = s_i$. We can write the PSP as

$$\begin{aligned}
(\textbf{PSP}) \quad \text{minimize} \quad & \sum_{i \in I, l \in \tau_A : t_l = s_i} (w_l/n - u_i x_i) + 1 - v && && (38)\\
\text{subject to} \quad & \sum_{j \in \delta_i} c_j x_j + c_i x_i \leq C, && i \in I, && (39)\\
& \sum_{j \in \delta_i^+} x_j - x_i + w_l \geq 0, && i \in I, l \in \tau_A : t_l = s_i, && (40)\\
& x_i \in \{0,1\}, && i \in I, && (41)\\
& w_l \in \{0,1\}, && l \in \tau_A, && (42)
\end{aligned}$$

which produces a server schedule with the smallest reduced cost. This optimization problem can be modeled on a graph $G = (V, A)$. The set of nodes $V = I \cup \{o^s, o^d\}$ comprises the VM requests, which are sorted in ascending order of their start times, and $o^s = 0$ and $o^d = n + 1$ are the source and sink nodes, respectively. Therefore, in this topologically sorted node set, each node $i \in V \backslash \{o^s, o^d\}$ coincides with the start time s_i of the corresponding request, which is also associated with the end time e_i. The arc set A is composed of (o^s, j) for $j \in V \backslash \{o^s, o^d\}$, (j, o^d) for $j \in V \backslash o^s$ and pairs of nodes (i, j) for $i, j \in V \backslash \{o^s, o^d\}$ and $i < j$. Each arc (i, j) is associated with a parameter $d_{ij} = -u_j$ for $i, j \in V \backslash \{o^d\}$, and $d_{io^d} = 0$ for $i \in V \backslash \{o^s, o^d\}$.

The optimal solution of PSP can be achieved by solving a shortest path problem with resource constraints on G for which we design a labeling algorithm (Irnich and Desaulniers, 2005). This algorithm is based on generating paths, or schedules, by processing partial paths on the nodes V, imposing constraints (39) along the path. To that end, we define a label X_p^i associated with a partial path p from o^d to node i. We define five resources that are attached to this label, namely

R_L^p, R_P^p, R_A^p, R_E^p and R_C^p. While R_P^p and R_A^p keep track of the visited nodes (requests) and the set of active requests on p ($R_A^p \subseteq \delta_i \cup \{i\}$), respectively, R_L^p represents the load of active requests of partial path p, hence can be defined as $R_L^p = \sum_{j \in R_A^p} c_j x_j$. R_E^p is the end time of active requests, i.e., the time the server becomes idle. This resource accelerates the algorithm by facilitating the calculation of the load on the server and of the number of fire-ups. Finally, R_C^p denotes the reduced cost of partial path p, as defined in (38).

The labeling algorithm starts with a null label at o^s whose resources are initialized as zero and empty sets except that $R_C^p = 1 - v$. In the rest of the algorithm, the nodes corresponding to I that are sorted in topological order of V are treated sequentially.

At step i of the algorithm, all the partial paths, and their related labels, on the node in the i^{th} position are extended to all the nodes in the j^{th} position where $j > i$. The extension of a label X_p^i associated with path p on node i to j to form a new label X_q^j related with path q involves updating the values of the resource with the addition of the new node j, which is trivial for $R_P^q = R_P^p \cup \{j\}$. We first check whether $R_E^p < s_j$ holds, in which case $R_E^q = e_j$, $R_A^q = \{j\}$, $R_L^q = c_j$ and $R_C^q = R_C^p + d_{ij} + 1/n$ due to the fire-up of the idle server with the start of j. Otherwise, $R_C^q = R_C^p + d_{ij}$, $R_A^q = R_A^p \cap \delta_j \cup \{j\}$, which induces $R_L^q = \sum_{j \in R_A^q} c_j x_j$, and $R_E^q = e_j$ if $e_j > R_E^p$, and $R_E^q = R_E^p$, otherwise. The extension of the label is feasibly effectuated only if $R_L^q \leq C$. The labels accumulated at the end of the algorithm on o^d are the feasible server schedules and, depending on how the column generation algorithm is implemented, one or more labels with $R_C^q < 0$ are added to the RMP.

The efficiency of the labeling algorithm hinges upon the effectiveness of the elimination of the labels that are not Pareto optimal. This is achieved by the dominance rule, whose validity is proved below. This rule is similar to that used in pick-up and delivery vehicle routing problem with time windows (Dumas et al., 1991).

Proposition 4.2. *Label X_p^i on node i dominates another label X_q^i on this node, if $R_C^p \leq R_C^q$, $R_E^p = R_E^q$ and $R_A^p \subseteq R_A^q$.*

Proof. To prove this proposition, we show that any feasible extension of X_q^i from node i to o^d is also feasible for X_p^i, and in all these extensions, X_q^i never attains smaller reduced cost than X_p^i. First, for an extension of these labels to node $j > i$, which satisfy $s_j > R_E^p = R_E^q$, both X_p^i and X_q^i have the same load and end time as j becomes the only active request. Hence, all partial paths from j to o^d attain the same values of the resources. Secondly, due to $R_A^p \subseteq R_A^q$ and $R_E^p = R_E^q$ as given in the proposition, $R_L^p \leq R_L^q$ also holds so that any node $j > i$, which satisfy $s_j \leq R_E^p = R_E^q$, that is visited by X_q^i is also available for X_p^i without incurring a fire-up cost. Therefore, in neither case, the extension of X_q^i can attain a smaller reduced cost than that of X_p^i since $R_{\mathcal{C}}^p \leq R_{\mathcal{C}}^q$. $\square$

Implementation Details

In order to accelerate the solution of the labeling algorithm, we aim to eliminate any unpromising partial path as early as possible in the iterations. A partial path p on a node, say i, whose best extension to o^d does not attain a negative reduced cost, can be eliminated without compromising on the optimality of the algorithm. This best possible extension of X_p^i can be obtained by finding a lower bound on its reduced cost at completion. To that end, at the outset of the labeling algorithm, we find the maximum total dual values of the nodes in a reverse path from o^d to each node. We denote the maximum total dual value from o^d to a node i by $\theta_i = \sum_{j=i}^{n} max(u_j, 0)$. Note that θ_i from o^d to i does not include those nodes j for which $u_j < 0$. Therefore, if extending a label, say X_p^i, associated with partial path p on node i to node j renders $R_{\mathcal{C}}^p - \theta_j \geq 0$, then the extension of this partial path to j is not carried out. This is a loose lower-bound as the load on the reverse path, which is not monotonically increasing on the reverse path, is not considered for the speed-up of this subprocedure. However, towards the end of the column generation algorithm, this bound leads to fast termination of the algorithm.

We also show through the following proposition that some extensions of partial paths can be eliminated without changing the optimal solution of the labeling algorithm.

Proposition 4.3. *Extension of label X_p^i associated with partial path p to a node $j \in V \backslash \{o^s, o^d\}$ can be disregarded, if $u_j < 0$ and $R_E^p \geq e_j$.*

Proof. Extension of X_p^i to j forms partial path q with $R_{\mathcal{C}}^q = R_{\mathcal{C}}^p + d_{ij} = R_{\mathcal{C}}^p - u_j > R_{\mathcal{C}}^p$, which does not trigger a fire-up since $R_E^p \geq e_j > s_j$. All the feasible extensions of X_q^j to o^d are also feasible for X_p^i since $R_L^q \geq R_L^p$ and $R_A^q = R_A^p \cup \{j\}$. In all feasible extensions where X_q^i does not incur a fire-up cost, X_p^i follows suit due to the given condition that $R_E^p = R_E^q \geq e_j$. $\square$

Invoking the labeling algorithm at each iteration leads to prolonged solution times due to its complexity. In order to reach the optimal LP relaxation solution, solving the labeling algorithm is imperative, though this can be deferred to the point in the algorithm when a heuristic version of the labeling algorithm fails to generate a negative reduced cost schedule. This heuristic labeling algorithm is predicated on a relaxation of the dominance rule, which accelerates the solution of PSP at the expense of eliminating some of the promising schedules. In particular, we impose the following relaxed dominance rule: Label X_p^i on node i dominates another label X_q^i on this node if $R_C^p \leq R_C^q$ and $R_L^p \leq R_L^q$. We have observed in the computational experiments that the solution time of the PSP has reduced substantially with this relaxation, and the number of times the exact labeling is invoked to prove optimality has been kept considerably low.

As alluded to previously, the labeling algorithm is capable of producing many schedules as candidates to be added to RMP. The choice of how many of these schedules shall be added at each iteration has an impact on the performance of the algorithm. We store the schedules (columns) with negative reduced cost on o^d at the end of the labeling algorithm in a column pool so that it can be searched for a negative reduced cost schedule before the PSP is called. If it does not contain such a schedule, then the pool is cleared and then refilled again after the solution of the PSP. Otherwise, we add the schedule with the most negative reduced cost to the RMP.

5. Computational Experiments

This section describes the computational experiments that we have performed to assess the performance of our upper and lower bounding methods. The bounding methods are implemented in C++ and run on an Intel Core i9-8950HK CPU (2.9 GHz) with 32 GB RAM. CPLEX 12.9 was used as a mixed integer linear programming (MILP) solver. In all experiments, a limit of 30 minutes is used unless otherwise indicated. In addition, the recovery algorithm is terminated if the solution does not change within 20 iterations.

5.1. Experimental Setup

Our experimental design is based on various factors, such as the number of VM requests (n), the length of the VM arrival period, the duration of the VM requests and the requested capacities (c_i). In our numerical experiments, we consider various problem sizes with $n = \{50, 100, 150, 200, 500, 1000\}$. The length of the arrival period is defined with respect to the arrival times of the VM requests. Let $\bar{s}$ denote the length of the arrival period. Then, we have $\max_{i \in I} s_i \leq \bar{s}$. $\bar{s}$ is adjusted according to the total number of VM requests. We set $\bar{s} \in \{n, 1.2n\}$ to represent tight and relaxed arrival schedule. For each VM $i \in I$, the start and end times are randomly generated. The start time, s_i, of VM i is a random integer in interval $[0, \bar{s}]$. In order to have a uniform VM load throughout the planning horizon, we divide the planning horizon into intervals of length 50 and generate a similar number of VM requests for each interval.

We test the proposed methods with respect to the length of requested time interval which can be considered as the duration of a VM request. We define two sets for VM duration, denoted by d_S and d_L, which are randomly generated in intervals $[10, 30]$ and $[20, 60]$, respectively. In order to generate the requested capacity for VM request i (c_i), we define two parameters, namely c_{min} and c_{max} where c_{min} is the minimum capacity demand and c_{max} is the maximum requested capacity. Then, we can generate integer capacity request for VM i randomly between c_{min} and c_{max}. In our experiments, we set $c_{min} = 25$, and test the models for varying capacity requirements by changing $c_{max} \in \{50, 75\}$. These two instances are denoted by c_L and c_H, respectively. In all of our numerical experiments, we assume that the servers are identical and the capacity is set to $C = 100$. We label our test problems by using all combinations of these parameters. For each combination, five instances were generated, totaling 240 instances. The following notation is used to denote the solution methods tested in the numerical experiments: (i) CG-LB/UB: Lower bound (LB) and upper bound (UB) obtained from column generation algorithm; (ii) M1-OPT: Optimal solution of mixed integer programming model $M1$; (iii) M1-LB: Optimal solution of the LP relaxation of model $M1$ by CPLEX; (iv) RMAT: Recovery heuristic; (v) CLH: Constructive look-ahead heuristic; (vi) BF: Best Fit heuristic; (vii) SMM: Server minimization model given by (1)-(4).

5.2. Numerical Results for Moderate Size Problems

In this section, we present and discuss our results from the experiments carried out. Table 1 reports the results of CG, RMAT, CLH, the solution of the exact model M1 (M1-OPT) and its LP relaxation (M1-LB) by CPLEX for moderate size problems with $n = \{50, 100, 150, 200\}$. The individual results for each test instance are given in Tables 5 - 8 in Appendix. In these experiments, we set the maximum neighbourhood size to 50 for RMAT. We note that the alternative formulation M2 was solely proposed to demonstrate the derivation of MP through decomposition. Furthermore, it attains inferior lower bounds compared to M1, as proved in Proposition 4.1, so that it is not considered in the computational experiments. The first four columns in Table 1 show the characteristics of the test instances. The next three columns present the comparative measures for CG, namely the average CPU time (in seconds) to find the optimal LP relaxation of MP, the number of instances for which CG solves the LP relaxation of MP within the time limit, the average lower and upper bound percentage gaps with the optimal solution. Upper bound optimality gaps are given in parentheses. As pointed out previously, the column generation algorithm is initialized with the schedules produced by RMAT so that the objective value found by CG-UB is always at least as good as that of RMAT. Therefore, we only report the average CG-UB gap when its value is better than the one provided by RMAT. The next two columns present the measures for M1-LB corresponding to the average CPU time and the average percentage gap with the optimal solution. The next two columns give the statistics for M1-OPT, namely average CPU time and the number of instances out of five M1-OPT solved to optimality by MILP solver. We note that the solution of CLH is used as an upper bound to limit the set of physical servers, K, in model M1. The last six columns present the comparison measures for RMAT such as the average CPU time, the number of instances out of five for which RMAT finds the optimal solution, the average percentage gaps with M1-OPT, M1-LB, CLH and CG-LB. A noteworthy observation is that the optimality of RMAT solution can be identified by comparing it with the M1-OPT, M1-LB and CG-LB solutions reported in Tables 5 - 8 in Appendix. For instance, both CG-LB and RMAT obtain the objective value of 16.120 in the fifth instance of test $(150, d_S, c_H)$ with $\bar{s} = 150$ in Table 7, while M1-OPT could not find the optimal solution within 30 minutes and terminates with the objective value of 16.173.

Comparing the percentage gaps under this setup, we observe that RMAT performs well and solves most of the problems to optimality. Although the percentage gap between RMAT and M1-OPT increases with the increase in problem size, in the worst-case, it is less than 3.75%. When we examine all individual test results presented in Tables 5 - 8, we observe that the average gap between the objective values obtained by RMAT and M1-OPT over the instances solved to optimality by the latter is less than two percent. On the other hand, the upper bound obtained by CG is generally the same with the one obtained by RMAT. As can be seen in Table 1, the CG upper bound improves the average optimality gap of the RMAT in one of the instances. More detailed comparison of the upper bounds are presented in Tables 5 - 8 in Appendix. When we compare the CLH and RMAT, we observe that RMAT improves the VM schedule obtained by CLH significantly.

When we look into the optimality gap of lower bounding methods, we observe that CG-LB performs quite well when the requested VM capacities are high. The optimality gap increases when the requested capacities are low and the duration of VM requests is short (see the rows corresponding to d_S and c_L in Table 1). Although the average optimality gap can be over 4% in some cases, the optimal objective values of CG-LB and M1-OPT is actually very close. When we examine the individual test results given by Tables 5 - 8, we observe that the difference between the optimal objective values of these two methods is less than two. On the other hand, relaxation lower bound M1-LB is tighter when the requested VM capacities are low (see the rows corresponding to c_L in Table 1). This behaviour can be attributed to the impact of symmetry breaking constraints defined for model M1. We observe that the optimal number of servers is closer to the lower bound on the number of servers given by equation (19) when the requested VM capacities are low.

As expected, the CPU time of the methods is highly dependent on the instance characteristics. As the number of VM requests increases, the solution time of CG, M1-OPT and RMAT increases. Table 1 shows that M1-OPT rarely obtains the optimal solution within the specified time limits when the number of VM requests is higher than 100. The solution time is also significantly dependent on the duration of the VM requests. As the durations of the VM requests increase, the number of active VM requests at each time period increases and the problem becomes more dense.

Table 1: Computational results for the test problems for $n = 50, 100, 150, 200$

Instances				CG			M1-LB		M1-OPT		Recovery Mat (RMAT)					
n	$\bar{s}$	d_i	c_{max}	Time	Succ	Gap(UBGap)	Time	Gap	Time	Opt	Time	Opt	M1-OPT Gap	M1-LB Gap	CLH Gap	CG Gap
50	50	d_S	c_L	7.0	5	4.21	0.1	4.80	12.8	5	8.3	5	0.0	4.80	9.30	4.21
			c_H	0.8	5	0.70	0.1	10.26	6.7	5	12.5	5	0.0	10.26	12.64	0.70
		d_L	c_L	5.6	5	1.94 (0.0)	0.1	0.07	159.4	2	39.9	1	3.33	5.70	6.19	3.90
			c_H	0.2	5	0.73	0.1	10.38	622.3	5	82.7	5	0.0	10.38	5.66	0.73
	60	d_S	c_L	5.6	5	4.05	0.1	2.67	121.1	5	5.5	5	0.0	2.67	6.79	4.05
			c_H	1.8	5	2.97	0.1	15.57	16.9	5	15.2	4	0.03	15.60	3.97	3.01
		d_L	c_L	3.6	5	2.60	0.1	4.65	767.7	3	37.4	3	0.0	5.77	7.39	3.59
			c_H	0.4	5	0.67	0.1	12.10	80.0	4	40.4	4	0.0	10.80	7.09	1.04
100	100	d_S	c_L	192.2	5	4.21	0.1	5.33	16.1	4	51.9	4	0.0	6.09	6.97	4.91
			c_H	22.0	5	0.65	0.1	20.07	253.2	5	57.0	4	0.01	20.08	2.92	0.66
		d_L	c_L	204.8	5	-	0.2	-	1800	-	58.9	-	-	5.92	4.35	6.32
			c_H	16.0	5	0.0	0.2	4.35	292.0	1	155.2	1	0.0	9.70	8.25	2.22
	120	d_S	c_L	683.2	5	3.59	0.1	0.07	73.4	3	41.1	2	3.71	6.52	4.15	7.89
			c_H	50.2	5	0.99	0.1	15.08	999.7	4	79.6	4	0.0	13.72	3.07	1.53
		d_L	c_L	133.6	5	-	0.1	-	1800	-	19.2	-	-	9.67	3.54	7.52
			c_H	19.7	5	0.0	0.2	13.74	1800	-	190.6	1	-	8.70	5.34	2.57
150	150	d_S	c_L	1236	2	3.72	0.1	0.04	212.7	5	17.6	3	2.01	2.05	7.09	3.72
			c_H	480.6	5	0.0	0.2	17.23	986.5	1	211.6	2	0.0	13.67	4.99	1.78
		d_L	c_L	412.4	5	-	0.2	-	1800	-	46.5	-	-	5.29	4.87	4.51
			c_H	144.5	5	-	0.3	-	1800	-	115.8	-	-	10.61	3.13	3.93
	180	d_S	c_L	697.0	1	0.0	0.1	4.76	108.8	5	13.7	4	2.22	7.26	4.06	0.0
			c_H	806.0	5	1.86	0.2	13.52	1233.8	3	69.6	2	0.02	13.14	8.51	1.87
		d_L	c_L	1519.0	2	-	0.2	-	1800	-	37.4	-	-	9.28	2.40	9.39
			c_H	218.7	5	0.0	0.3	14.46	1800	-	218.2	-	-	16.49	2.71	2.46
200	200	d_S	c_L	1800	0	-	0.1	0.03	747.5	1	45.6	1	0.0	9.20	4.82	-
			c_H	1800	0	-	0.2	-	1800	-	139.7	-	-	12.76	5.14	-
		d_L	c_L	1800	0	-	0.3	-	1800	-	23.8	-	-	10.90	0.14	-
			c_H	817.7	3	0.0	0.4	20.84	1800	-	177.4	1	-	13.91	3.05	3.33
	240	d_S	c_L	1800	0	-	0.1	6.70	407.3	5	28.0	2	0.04	7.73	7.47	-
			c_H	1800	0	-	0.2	-	1800	-	124.2	-	-	19.11	4.37	-
		d_L	c_L	1800	0	-	0.2	-	1800	-	28.4	-	-	11.77	2.17	-
			c_H	1044.0	5	-	0.4	-	1800	-	228.9	-	-	13.79	6.86	1.39

5.3. Numerical Results for Large Instances

In this section, we report our results for larger instances with two different experiments. In the first experiment, we generate large-scale instances with $n = \{500, 1000\}$ synthetically as described in the experimental setup. On the other hand, our second experiment is performed on a real-world data set recorded by Google in 2010 (Hellerstein, 2010).

In our first experiment, we compare the performances of RMAT, CLH and BF only since CG and M1-OPT fail to solve these instances to optimality. Table 2 presents the average CPU times, average number of physical servers and the average number of fire-ups obtained by these solution methods with respect to various test instances. As depicted in Table 2, RMAT performs better than CLH and BF and significantly improves the average number of fire-ups and the average number of required physical servers. Although CLH and BF aim to minimize the number of fire-ups, they fail to schedule the VM requests efficiently. The differences between the obtained number of fire-ups are more striking when the requested capacities are higher and the durations of VM requests are short. As the requested VM durations decrease, the overlaps between VM requests also decrease which may increase the number of fire-ups (see the rows corresponding to d_S in Table 2). When we look into the computational times, we observe that the CPU time of RMAT is generally less than ten minutes.

In the second experiment, we test the performances of the models on a real-world data set. We extracted VM request data (approximate arrival and departure times and memory requirements) recorded by Google in 2010 (Hellerstein, 2010). In order to mask the private information, VM memory requirements were normalized to $0-1$ scale in the given data set. Therefore, we set the capacity of physical servers to 1 for this experiment. To generate test instances, we uniformly sample VM requests arrived during the recorded time period. For each test problem, five instances are sampled, totaling 25 instances. Since CG cannot be solved to optimality for these instances, we compare the performances of RMAT, M1-OPT, M1-LB, CLH and BF on the test problems with $n = \{1000, 2000, 3000, 4000, 5000\}$. We set the size of the neighbourhood for RMAT to 70. Table 3 reports the CPU times, the average number of physical servers and the average number of fire-ups obtained by these solution methods. We also report some additional statistics for RMAT. The column "M1-OPT Gap" presents the average percentage gaps between the optimal number of physical servers and fire-ups obtained by RMAT and M1. If the optimal solution could not be found within the time limit, the best feasible solution obtained is used to compute the percentage gaps which are reported in parentheses. Similarly, the column "M1-LB Gap" shows the average

Table 2: Computational results for the test problems for $n = 500, 1000$

Instances				RMAT			CLH			BF		
n	$\bar{s}$	d_i	c_{max}	Time	z	w	Time	z	w	Time	z	w
500	500	d_S	c_L	53.82	12.8	16.0	0.52	13.2	48.0	0.03	13.2	60.8
			c_H	332.51	18.6	46.0	0.49	18.8	107.2	0.03	19.6	131.6
		d_L	c_L	54.82	22.0	23.6	0.45	22.4	43.4	0.03	22.4	50.8
			c_H	258.19	30.0	44.6	0.45	30.4	92.8	0.04	30.6	111.2
	600	d_S	c_L	89.66	11.6	16.0	0.32	12.2	50.2	0.03	11.8	62.0
			c_H	263.52	15.6	52.8	0.33	15.8	122.4	0.03	16.2	140.2
		d_L	c_L	85.28	19.2	20.0	0.43	19.4	45.0	0.03	19.6	51.8
			c_H	210.87	26.2	46.0	0.42	26.4	97.2	0.03	26.6	112.0
1000	1000	d_S	c_L	63.21	13.0	22.2	1.10	13.4	92.0	0.08	13.6	118.0
			c_H	477.61	20.6	97.4	1.04	21.0	228.4	0.10	21.2	265.2
		d_L	c_L	76.86	22.0	24.4	1.43	22.0	72.4	0.11	22.0	80.4
			c_H	665.79	32.6	85.0	1.02	33.2	186.8	0.12	33.8	216.6
	1200	d_S	c_L	60.50	11.8	25.8	1.06	11.8	100.6	0.09	12.0	121.2
			c_H	514.73	17.6	105.2	1.07	18.0	240.8	0.09	18.2	284.8
		d_L	c_L	82.60	19.4	22.8	1.33	19.4	76.20	0.10	19.6	87.2
			c_H	598.91	27.2	75.2	1.29	27.6	196.6	0.12	27.6	227.2

percentage gaps between the total number of physical servers and fire-ups obtained by RMAT and M1-LB. The individual results for each test instance are given by Table 9 in Appendix. Comparing the results in Table 1 with those in Table 3, we note that the real case instances are easier to solve. M1-OPT can be solved to optimality within 30 minutes time limit up to the problem size of 4000. In addition, LP relaxation bound for M1-OPT is very tight for all instances. When we analyse the data set, we observe that the optimal number of servers is generally equal to the bound given by the equation (19). Therefore, symmetry breaking constraints defined for model M1 improve the LP relaxation and help to solve the larger size instances to optimality. When we look into the performance of RMAT, we see that it gives the optimal solution for almost all cases. We can identify the optimal solutions for the test instances with size $n = \{3000, 4000, 5000\}$ by comparing the solutions of RMAT and M1-LB given in Table 9. For instance, both RMAT and M1-LB obtain the same number of servers in the first test case of $n = 5000$ in Table 9 and the difference in number of fire-ups is less than one. This shows that RMAT finds the optimal solution for this test case. When we analyse the results obtained by CLH and BF, we observe that these heuristics generally perform well. They find the optimal number of servers for most of the test instances as we can identify from Table 9.

Table 3: Computational results for the real-world data set

Instance Size	M1-LB			M1-OPT				RMAT					CLH			BF		
n	Time	z	w	Time	z	w	Opt	Time	z	w	M1-OPT Gap	M1-LB Gap	Time	z	w	Time	z	w
1000	0.6	12.6	12.1	4.1	12.6	12.6	5	3.7	12.6	12.6	0.0	2.0	0.8	12.6	14.0	0.1	12.6	14.2
2000	24.5	24.0	23.4	32.2	24.0	24.0	5	8.3	24.0	24.0	0.0	1.2	2.4	24.0	26.6	0.1	24.0	27.8
3000	150.8	35.6	34.9	437.8	35.6	35.8	4	20.5	35.6	35.6	0.0 (-1.4)	1.0	3.4	35.6	39.8	0.3	35.6	43.2
4000	240.9	47.4	46.9	458.4	47.6	48.0	3	48.6	47.6	47.6	0.0 (-1.0)	1.0	5.6	47.6	53.0	0.4	47.6	55.6
5000	971.5	60.2	59.8	1800	61	62.2	-	127.3	60.8	60.8	- (-1.2)	1.3	8.0	60.8	68.4	0.7	60.8	73.0

5.4. Single Objective vs. Multiple Objectives

In this section, we compare the performances of M1-OPT, RMAT and that of single objective model (1)-(4), denoted by SMM, in terms of the optimal number of required physical servers and fire-ups separately. Note that, in Table 4, we only report the average test results where both SMM and M1-OPT are solved to optimality. The first four columns in Table 4 show the characteristics of the test instances. The fifth column shows the number of test instances solved out of five within the specified time limit by SMM and M1-OPT. We note that the instances are solved by RMAT with the average time of 32.9 seconds, whereas in those parameter settings where the number of solved instances is smaller than five, SMM and/or M1-OPT fail to solve all instances to optimality within 30 minutes. The next columns present the comparison measures for each method, namely the average CPU time (in seconds) to solve the instances, the average number of physical servers and the average number of fire-ups. As Table 4 depicts, the optimal number of servers is equal for SMM and M1-OPT in all instances. By comparing the average number of fire-ups obtained by SMM and other methods, we observe that the fire-ups can be significantly reduced when it is included in the objective function. The improvements in the number of fire-ups are more significant as the problem size increases (see the rows corresponding to $n = 150, 200$ in Table 4). Thus, by considering the minimization of fire-ups as one of the objectives in the VMP problem, we can improve the VM assignment schedule and increase the effectiveness of the shared physical servers.

In order to evaluate the impact of coefficient γ on the optimal number of active servers and fire-ups, we repeat the numerical experiment by setting $\gamma = 1$. The average test results where both SMM and M1-OPT are solved to optimality are reported by Table 10 in Appendix. Comparing the results where both SMM and M1-OPT solved all five test cases in Table 4 with those in Table 10, we observe that the optimal number of active servers and fire-ups does not change when we level both objectives with $\gamma = 1$. However, we observe that the number of solved tests changes for M1-OPT when we adjust the value of coefficient γ.

Table 4: Average server and fire-ups

Instances				# Solved	SMM			M1-OPT			RMAT		
n	$\bar{s}$	d_i	c_{max}		Time	z	w	Time	z	w	Time	z	w
50	50	d_S	c_L	5	1.9	9.8	11.4	12.8	9.8	9.8	8.3	9.8	9.8
50	50	d_S	c_H	5	1.5	12.8	20.8	6.7	12.8	13.0	12.5	12.8	13.0
50	50	d_L	c_L	2	317.1	15.0	16.0	159.4	15.0	15.0	39.9	15.5	15.5
50	50	d_L	c_H	5	173.7	23.0	25.8	622.3	23.0	23.0	82.7	23.0	23.0
50	60	d_S	c_L	5	6.5	8.4	10.6	121.1	8.4	8.4	5.5	8.4	8.4
50	60	d_S	c_H	5	1.2	11.8	18.8	16.9	11.8	12.2	15.2	11.8	12.4
50	60	d_L	c_L	1	764.4	15.0	15.5	643.4	15.0	15.0	48.8	15.0	15.0
50	60	d_L	c_H	4	23.5	21.2	25.5	80.0	21.2	21.2	34.4	21.2	21.2
100	100	d_S	c_L	4	1.1	11.0	17.7	16.1	11.0	11.0	45.9	11.0	11.0
100	100	d_S	c_H	5	17.9	16.8	45.2	253.2	16.8	18.0	47.0	16.8	18.2
100	100	d_L	c_H	1	346.7	24.0	34.0	292.0	24.0	24.0	69.6	24.0	24.0
100	120	d_S	c_L	3	3.6	9.7	18.0	73.4	9.7	9.7	19.0	9.7	9.7
100	120	d_S	c_H	4	34.5	13.5	42.0	999.7	13.5	16.0	57.7	13.5	16.0
150	150	d_S	c_L	5	24.1	10.8	20.0	212.7	10.8	10.8	17.6	11.0	11.2
150	150	d_S	c_H	1	53.3	18.0	62.0	986.5	18.0	22.0	104.4	18.0	22.0
150	180	d_S	c_L	5	163.6	9.8	24.0	108.8	9.8	9.8	13.7	10.0	10.0
150	180	d_S	c_H	3	63.1	14.0	62.3	1233.8	14.0	19.7	62.2	14.0	20.0
200	200	d_S	c_L	1	178.3	12.0	32.0	747.5	12.0	12.0	22.1	12.0	12.0
200	240	d_S	c_L	5	56.5	10.6	39.4	407.3	10.6	11.0	28.0	10.6	11.8

6. Conclusion

In this paper, we looked into the assignment problem arising in cloud computing, namely the virtual machine placement problem. This problem has been studied extensively in the literature, however, we aimed to establish its position in the general combinatorial optimization literature by discussing its similarity to various prominent problems, the bin packing problem with time dimension being the most relevant. The major impact of this study is in the way the multi-objective optimization problem is modeled, where the usual bin packing objective minimizing the number of servers is augmented with the energy efficiency related objective of minimizing the number of fire-ups on the used servers.

In order to solve the model, we proposed a heuristic algorithm based on exact solutions of smaller problems defined on neighborhoods of interest iteratively. The performance of this heuristic has been proven to be superior, which is proved by the MIP solution and a lower bound obtained by our proposed column generation algorithm. Even though the VMP problem modeled with identical servers, the heuristic algorithm is capable of solving servers with variable capacities. However, the column generation algorithm must be modified to handle the heterogeneous case, which we plan to work as a future study.

We plan to extend our work presented here to the problems arising in various applications, e.g., scientific computing services where the computing requests are to be assigned to high-performance computers. This application brings about several other concerns regarding the priority, precedence and computational speed of the requests.

Acknowledgment

We appreciate the comments offered by the referees. Their comments certainly helped us improve this work considerably.

Appendix

Table 5: Computational results for each test instance ($n = 50$)

Instances	$\bar{s} = 50$					$\bar{s} = 60$				
(n, d_i, c_{max})	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH
(50, d_S, c_L)	8.798/9.180	9.166	9.180	9.180	10.200	7.905/8.160	8.146	8.160	8.160	9.180
	9.971/10.200	10.192	10.200	10.200	11.220	7.841/8.160	8.141	8.160	8.160	9.200
	8.670/9.180	8.160	9.180	9.180	10.200	8.415/9.180	8.150	9.180	9.180	9.200
	10.725/11.220	11.208	11.220	11.220	12.240	8.160/8.160	8.147	8.160	8.160	9.180
	9.818/10.200	9.175	10.200	10.200	11.220	8.840/9.180	9.165	9.180	9.180	9.200
(50, d_S, c_H)	12.240/12.240	11.220	12.240	12.240	15.340	11.730/12.240	10.199	12.240	12.240	12.300
	14.280/14.280	12.230	14.280	14.280	15.420	14.300/14.300	11.218	14.300	14.300	14.380
	13.280/13.280	12.234	13.280	13.280	15.340	10.740/11.240	10.199	11.240	11.240	12.320
	13.800/14.280	12.234	14.280	14.280	15.400	11.750/12.260	10.199	12.240	12.260	12.300
	11.220/11.220	11.202	11.220	11.220	13.260	10.030/10.200	10.187	10.200	10.200	11.300
(50, d_L, c_L)	16.065/16.320	15.288	16.320†	16.320	17.340	13.849/14.280	13.258	14.280†	14.280	15.300
	14.496/15.300	14.278	15.300†	15.300	16.320	15.810/16.320	15.292	16.320	16.320	18.360
	14.025/14.280	13.249	14.280†	14.280	15.300	14.663/15.300	14.268	15.300	15.300	16.320
	14.729/15.300	15.284	15.300	15.300	16.320	14.293/15.300	14.275	15.300†	15.300	16.320
	15.300/15.300	15.295	15.300	16.320	17.340	14.246/14.280	14.279	14.280	14.280	15.300
(50, d_L, c_H)	18.700/19.380	18.353	19.380	19.380	20.400	22.440/22.440	19.363	22.440	22.440	24.520
	22.440/22.440	21.401	22.440	22.440	23.460	18.870/19.380	18.351	19.380	19.380	20.400
	24.480/24.480	21.400	24.480	24.480	25.500	18.910/19.380	18.352	19.380†	19.380	21.420
	27.540/27.540	23.446	27.540	27.540	29.600	24.480/24.480	22.424	24.480	24.480	26.520
	23.460/23.460	21.402	23.460	23.460	25.500	20.400/20.400	17.330	20.400	20.400	21.420

† M1-OPT could not find the optimal solution within 30 minutes.

Table 6: Computational results for each test instance ($n = 100$)

Instances	$\bar{s} = 100$					$\bar{s} = 120$				
(n, d_i, c_{max})	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH
(100, d_S, c_L)	9.595/10.100	9.090	10.100	10.100	11.170	9.617/10.100	10.091	10.100	10.100	11.170
	10.942/11.110	11.101	11.110	11.110	12.180	10.352/11.110	10.091	11.110	11.110	11.170
	11.511/12.120	12.112	12.120	12.120	13.190	8.964/10.100	9.086	9.090	10.100	10.140
	11.254/12.120	11.106	12.120†	12.120	13.190	9.679/10.100	10.091	10.100	10.100	11.180
	10.605/11.110	10.095	11.110	11.110	11.140	9.176/10.100	9.086	10.100†	10.100	10.160
(100, d_S, c_H)	20.210/20.210	16.153	20.210	20.210	20.280	13.705/14.210	13.120	14.210†	14.210	14.340
	17.180/17.180	13.129	17.180	17.180	18.270	12.660/13.150	11.103	13.150	13.150	13.250
	15.180/15.180	13.128	15.180	15.180	16.290	15.165/15.170	13.127	15.170	15.170	17.340
	15.665/16.180	14.137	16.170	16.180	16.280	12.145/12.150	11.102	12.150	12.150	12.260
	16.160/16.160	14.130	16.160	16.160	16.300	14.165/14.170	12.125	14.170	14.170	14.210
(100, d_L, c_L)	17.107/18.180	17.167	19.190†	18.180	19.220	14.235/15.150	14.135	15.150†	15.150	15.180
	18.012/19.190	18.171	19.190†	19.190	19.190	14.764/16.160	14.136	16.160†	16.160	17.180
	17.128/18.180	17.166	18.180†	18.180	19.210	14.014/15.150	14.134	15.150†	15.150	15.170
	17.026/18.180	17.164	19.190†	18.180	19.230	16.413/17.170	16.152	17.170†	17.170	18.210
	16.218/17.170	16.156	18.180†	17.170	18.180	15.756/17.170	15.147	16.160†	17.170	18.220
(100, d_L, c_H)	22.220/23.230	22.216	24.250†	23.230	26.300	20.203/21.220	20.195	21.250†	21.220	22.310
	28.290/28.310	23.226	28.320†	28.310	30.400	25.270/25.270	22.218	26.330†	25.270	26.370
	24.240/24.240	23.229	24.240	24.240	26.270	20.537/21.210	20.194	21.210†	21.210	22.300
	22.220/23.240	21.209	24.240†	23.240	25.260	22.745/23.250	21.205	23.330†	23.250	25.340
	26.785/27.290	25.246	27.290†	27.290	29.340	21.715/22.220	20.194	22.220†	22.220	23.270

† M1-OPT could not find the optimal solution within 30 minutes.

Table 7: Computational results for each test instance ($n = 150$)

Instances	$\bar{s} = 150$					$\bar{s} = 180$				
(n, d_i, c_{max})	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH
(150, d_S, c_L)	- /10.067	10.064	10.067	10.067	11.147	10.067/10.067	10.062	10.067	10.067	11.160
	10.822/11.073	11.068	11.073	11.073	12.140	- /9.060	9.054	9.060	9.060	9.120
	- /11.067	10.060	10.067	11.073	11.120	- /10.067	8.053	9.060	10.067	10.107
	11.493/12.080	12.074	12.080	12.080	13.127	- /10.067	9.057	10.067	10.067	10.127
	- /11.080	11.070	11.073	11.080	12.120	- /11.073	11.068	11.073	11.073	12.147
(150, d_S, c_H)	18.146/18.147	15.114	18.147	18.147	19.247	13.683/14.180	13.122	14.180†	14.180	16.307
	17.680/18.187	16.117	18.187†	18.187	19.313	11.838/12.087	11.007	12.087	12.087	13.200
	17.160/17.167	16.101	17.160†	17.167	18.273	14.121/14.133	12.074	14.133†	14.133	16.313
	15.216/16.127	14.095	16.127†	16.127	16.240	14.650/15.153	13.102	15.153	15.153	15.260
	16.120/16.120	14.092	16.173†	16.120	17.267	15.146/15.160	13.095	15.153	15.160	16.300
(150, d_L, c_L)	18.624/19.127	18.119	20.133†	19.127	20.167	- /16.107	15.098	17.113†	16.107	17.153
	20.039/21.140	20.130	22.147†	21.140	22.167	- /17.120	15.100	18.120†	17.120	18.133
	19.756/21.140	20.127	21.140†	21.140	21.153	- /16.107	15.097	-*	16.107	16.127
	18.875/20.133	19.122	21.160†	20.133	21.153	15.690/17.120	16.101	17.140†	17.120	17.133
	19.001/19.127	18.120	20.133†	19.127	21.180	15.603/17.113	15.099	17.113†	17.113	17.133
(150, d_L, c_H)	23.661/25.173	23.152	26.227†	25.173	26.240	25.173/25.180	20.133	25.173†	25.180	26.267
	27.690/28.190	25.162	-*	28.193	28.267	20.983/22.153	20.129	22.247†	22.153	22.240
	27.704/29.200	26.169	30.380†	29.200	31.307	24.685/25.200	21.140	26.280†	25.200	26.327
	28.691/29.193	26.171	-*	29.193	29.280	24.186/24.193	21.136	24.207†	24.193	25.293
	23.171/24.167	22.141	-*	24.167	25.253	24.063/25.167	22.146	25.220†	25.167	25.267

† M1-OPT could not find the optimal solution within 30 minutes.

$*$ M1-OPT could not find a feasible solution within 30 minutes.

Table 8: Computational results for each test instance ($n = 200$)

Instances	$\bar{s} = 200$					$\bar{s} = 240$				
(n, d_i, c_{max})	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH	CG-LB/UB	M1-LB	M1-OPT	RMAT	CLH
(200, d_S, c_L)	- /12.060	11.054	12.060†	12.060	13.120	- /10.060	9.049	10.060	10.060	11.130
	- /12.060	12.056	12.060	12.060	13.145	- /10.050	9.045	10.050	10.050	11.140
	- /14.075	13.064	14.075†	14.075	15.125	- /11.055	11.050	11.055	11.060	12.140
	- /12.060	10.049	11.060†	12.060	12.130	- /10.060	9.045	10.050	10.060	10.130
	- /12.060	11.055	12.060†	12.060	12.090	- /12.065	12.057	12.060	12.065	13.165
(200, d_S, c_H)	- /17.125	15.084	18.175†	17.125	18.260	- /15.130	13.070	15.160†	15.130	17.275
	- /17.125	14.066	17.195†	17.125	17.275	- /15.125	12.056	14.160†	15.125	15.265
	- /17.105	15.071	17.130†	17.105	17.230	- /15.120	14.071	15.185†	15.120	16.260
	- /14.070	13.062	13.080†	14.070	14.190	- /15.175	13.107	15.190†	15.175	15.315
	- /15.100	14.069	15.135†	15.100	18.205	- /17.150	13.078	17.175†	17.150	17.250
(200, d_L, c_L)	- /21.105	18.089	21.110†	21.105	21.140	- /18.090	17.080	19.115†	18.090	20.135
	- /21.105	19.092	21.110†	21.105	21.125	- /16.085	14.069	16.080†	16.085	16.100
	- /21.110	19.092	21.105†	21.110	21.120	- /18.090	16.075	18.090†	18.090	18.115
	- /19.095	18.086	-*	19.095	19.120	- /19.100	17.083	19.095†	19.100	19.145
	- /20.100	18.087	-*	20.100	20.145	- /16.080	14.069	16.085†	16.080	16.115
(200, d_L, c_H)	- /26.155	24.118	-*	26.155	26.260	23.627/24.135	21.105	-*	24.135	25.245
	29.145/29.145	24.118	-*	29.145	30.230	21.105/21.110	19.095	23.210†	21.110	23.240
	24.623/26.130	23.113	-*	26.130	27.260	24.137/24.155	21.103	25.225†	24.155	26.280
	- /29.150	25.122	30.265†	29.150	30.250	24.131/24.145	21.102	-*	24.145	25.235
	29.041/30.165	27.132	31.240†	30.165	31.245	22.115/23.140	20.097	25.325†	23.140	25.240

† M1-OPT could not find the optimal solution within 30 minutes.

$*$ M1-OPT could not find a feasible solution within 30 minutes.

Table 9: Computational results for real world application

Instance Size	M1-LB		M1-OPT		RMAT		CLH		BF	
n	z	w	z	w	z	w	z	w	z	w
1000	13	12.1	13	13	13	13	13	15	13	15
	12	11.8	12	12	12	12	12	14	12	14
	12	11.7	12	12	12	12	12	12	12	13
	13	12.2	13	13	13	13	13	14	13	14
	13	12.7	13	13	13	13	13	15	13	15
2000	23	22.8	23	23	23	23	23	25	23	25
	24	23.8	24	24	24	24	24	25	24	26
	24	23.5	24	24	24	24	24	26	24	28
	24	23.1	24	24	24	24	24	26	24	27
	25	24.1	25	25	25	25	25	31	25	33
3000	36	35.0	36	36	36	36	36	41	36	41
	35	34.2	35†	36†	35	35	35	38	35	39
	35	34.5	35	35	35	35	35	42	35	47
	36	35.4	36	36	36	36	36	38	36	47
	36	35.5	36	36	36	36	36	40	36	42
4000	47	46.3	47	47	47	47	47	53	47	59
	47	46.9	48†	48†	48	48	48	51	48	51
	47	46.3	47	47	47	47	47	50	47	57
	49	48.1	49†	51†	49	49	49	58	49	58
	47	46.7	47	47	47	47	47	53	47	53
5000	61	60.1	61†	63†	61	61	61	68	61	73
	60	59.7	61†	63†	61	61	61	70	61	76
	61	60.9	62†	62†	62	62	62	68	62	73
	59	58.8	60†	61†	60	60	60	69	60	73
	60	59.3	60†	62†	60	60	60	67	60	70

† M1-OPT could not find the optimal solution within 30 minutes.

Table 10: Average server and fire-ups for $\gamma = 1$

Instances				Num of	SMM			M1-OPT			RMAT		
n	$\bar{s}$	d_i	c_{max}	Tests	Time	z	w	Time	z	w	Time	z	w
50	50	d_S	c_L	5	1.9	9.8	11.4	36.6	9.8	9.8	35.7	9.8	9.8
50	50	d_S	c_H	5	1.5	12.8	20.8	35.7	12.8	13.0	27.5	12.8	13.0
50	50	d_L	c_L	1	2.9	15.0	16.0	69.0	15.0	15.0	65.8	15.0	15.0
50	50	d_L	c_H	5	173.7	23.0	25.8	481.5	23.0	23.0	125.6	23.0	23.0
50	60	d_S	c_L	4	3.5	8.2	9.7	10.6	8.2	8.2	13.9	8.2	8.2
50	60	d_S	c_H	4	4.1	11.2	17.2	65.4	11.2	11.5	15.0	11.2	11.5
50	60	d_L	c_L	2	855.1	14.0	14.5	113.5	14.0	14.0	29.6	14.0	14.0
50	60	d_L	c_H	4	23.5	21.2	25.5	242.7	21.2	21.2	75.5	21.2	21.2
100	100	d_S	c_L	3	8.2	11.0	18.0	108.3	11.0	11.0	11.1	11.0	11.0
100	100	d_S	c_H	5	17.9	16.8	45.2	479.3	16.8	18.0	42.5	16.8	18.4
100	120	d_S	c_L	3	3.6	9.7	18.0	225.6	9.7	9.7	53.8	9.7	9.7
100	120	d_S	c_H	5	61.5	13.6	40.6	641.4	13.6	17.0	64.9	13.6	17.2
150	150	d_S	c_L	4	16.0	10.7	18.7	374.3	10.7	10.7	74.3	11.0	11.0
150	180	d_S	c_L	5	163.6	9.8	24.0	290.0	9.8	9.8	16.3	9.8	9.8
150	180	d_S	c_H	2	21.1	13.5	58.5	509.3	13.5	18.0	33.0	13.5	18.5
200	200	d_S	c_L	1	415.1	12.0	38.0	1771.7	12.0	12.0	150.0	12.0	12.0
200	240	d_S	c_L	5	56.5	10.6	39.4	180.1	10.6	11.0	63.4	10.6	11.0

References

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance P. Branch-and-price: column generation for solving huge integer programs. Operations Research 1998;46(3):316–29.

Beloglazov A. Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing. Ph.D. thesis; Department of Computing and Information Systems, The Univeristy of Melbourne; 2013.

Calcavecchia NM, Biran O, Hadad E, Moatti Y. VM placement strategies for cloud scenarios. In: IEEE Fifth International Conference on Cloud Computing. 2012. p. 852–9.

Caprara A, Furini F, Malaguti E. Uncommon dantzig-wolfe reformulation for the temporal knapsack problem. INFORMS Journal on Computing 2013;25(3):560–71.

Caprara A, Furini F, Malaguti E, Traversi E. Solving the temporal knapsack problem via recursive dantzig–wolfe reformulation. Information Processing Letters 2016;116(5):379–86.

Cauwer MD, Mehta D, O'Sullivan B. The temporal bin packing problem: An application to workload management in data centres. In: 28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016. 2016. p. 157–64. doi:10.1109/ICTAI.2016.0033.

Chaisiri S, Lee B, Niyato D. Optimal virtual machine placement across multiple cloud providers. In: 2009 IEEE Asia-Pacific Services Computing Conference (APSCC). 2009. p. 103–10.

Cohen MC, Keller PW, Mirrokni V, Zadimoghaddam M. Overcommitment in cloud services - bin packing with chance constraints. 2016. Available at SSRN 2822188.

Dantzig G, Wolfe P. Decomposition principle for linear programs. Operations Research 1960;8(1):101–11.

De Carvalho JV. LP models for bin packing and cutting stock problems. European Journal of Operational Research 2002;141(2):253–73.

Dell'Amico M, Fischetti M, Toth P. Heuristic algorithms for the multiple depot vehicle scheduling problem. Management Science 1993;39(1):115–25.

Dell'Amico M, Furini F, Iori M. A branch-and-price algorithm for the temporal bin packing problem. arXiv preprint arXiv:190204925 2019;.

Delorme M, Iori M, Martello S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research 2016;255(1):1 – 20.

Di S, Kondo D, Cappello F. Characterizing cloud applications on a google data center. In: 2013 42nd International Conference on Parallel Processing. 2013. p. 468–73.

Dong J, Jin X, Wang H, Li Y, Zhang P, Cheng S. Energy-saving virtual machine placement in cloud data centers. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. 2013. p. 618624.

Dumas Y, Desrosiers J, Soumis F. The pickup and delivery problem with time windows. European journal of operational research 1991;54(1):7–22.

Fan L, Gu C, Qiao L, Wu W, Huang H. Greensleep: A multi-sleep modes based scheduling of servers for cloud data center. In: 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM). 2017. p. 368–75.

Ferland JA, Michelon P. The vehicle scheduling problem with multiple vehicle types. Journal of the Operational Research Society 1988;39(6):577–83.

Furini F, Shen X. Matheuristics for the temporal bin packing problem. In: Recent Developments in Metaheuristics. Springer; 2018. p. 333–45.

Gao Y, Guan H, Qi Z, Hou Y, Liu L. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. Journal of Computer and System Sciences 2013;:1230–42.

Glover F. Tabu search-part I. ORSA Journal on Computing 1989;1(3):190–206.

Gu C, Li Z, Huang H, Jia X. Energy efficient scheduling of servers with multi-sleep modes for cloud data center. IEEE Transactions on Cloud Computing, In Press 2018;:1–14.

Hellerstein JL. Google cluster data. 2010. Website.

Irnich S, Desaulniers G. Shortest path problems with resource constraints. In: Column generation. Springer; 2005. p. 33–65.

Jiang D, Huang P, Lin P, Jiang J. Energy efficient VM placement heuristic algorithms comparison for cloud with multidimensional resources. Information Computing and Applications 2012;:413–20.

Liu X, Zhan Z, Deng JD, Li Y, Gu T, Zhang J. An energy efficient ant colony system for virtual machine placement in cloud computing. IEEE Transactions on Evolutionary Computation 2018;22(1):113–28.

Mangoubi R, Mathaisel DF. Optimizing gate assignments at airport terminals. Transportation Science 1985;19(2):173–88.

Martello S, Toth P. Bin-packing problem. Knapsack problems: Algorithms and computer implementations 1990;:221–45.

Pires FL, Barn B. A virtual machine placement taxonomy. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2015. p. 159–68.

Speitkamp B, Bichler M. A mathematical programming apporach for server consolidation problems in virtualized data centers. IEEE Transactions on Services Computing 2010;3:266–78.

Wang G, Ben-Ameur W, Ouorou A. A lagrange decomposition based branch and bound algorithm for the optimal mapping of cloud virtual machines. European Journal of Operational Research 2019;276(1):28 – 39.

Wu Y, Tang M, Fraser W. A simulated annealing algorithm for energy efficient virtual machine placement. In: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2012. p. 1245–50.

Xie R, Jia X, Yang K, Zhang B. Energy saving virtual machine allocation in cloud computing. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops. 2013. p. 132–7.