Bin Packing Problem with Time Dimension: 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

[*] 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)

from the 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. In this paper, we consider the temporal extension of the bin packing problem, where the tasks are to be performed in fixed start and end times during a planning period. Our extension is motivated by cloud computing and concerns the most important operational cost in these services, the energy consumption. However, it is important to note that in other bin packing applications, where tasks should be packed according to a predefined schedule, the proposed approaches in the present paper can be readily applied.

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 are actual computers with different operating systems and they are allocated on 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. 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). Minimizing the number of active servers is one effective way of cutting down the energy consumption (Beloglazov, 2013; Jennings and Stadler, 2015). In the literature, there are several variations of this problem and, in general, the assignment decision does not take into consideration the temporal aspect of the VM requests (Chaisiri et al., 2009; Wu et al., 2012; Gao et al., 2013; Liu et al., 2018). Most of these 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). On the other hand, few studies reflect the time dimensions in their models, and tailor heuristics for the solution of this problem (Speitkamp and

Bichler, 2010; Calcavecchia et al., 2012; Xie et al., 2013; Puschel et al., 2015). Several algorithms have been proposed for this challenging problem and most of these are based on best fit and first fit heuristics (Lee et al., 2011). 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. 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 can be deemed as a bin packing problem with time dimension. 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. We show that the optimal solution of this model does not compromise on minimizing the number of servers while minimizing the number of fire-ups. 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. 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 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, which we assume to be larger than or equal to one, i.e. $c_i \geq 1$. Moreover, each VM resides in the system for the duration within the requested time interval $[s_i, e_i]$, where s_i and e_i are the start and the end times of request $i \in I$, respectively. 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 constraints are 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 $t_l = \{0, \cdots, T\}$, $l \in \tau$. 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. This model is tantamount to a 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 only if physical server k is switched 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 $l \in \tau$. 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 $w_l^k = 1$. Then, the total number of fire-ups for server k is given by $\sum_{l\in\tau} w_l^k = \sum_{l\in\tau} \max\{y_l^k - y_{l-1}^k, 0\}$. We assume that if a new request j arrives at the departure time of another request i, which is the only request on server k; *i.e.* $s_j = e_i$, then the server stays switched on. Consequently, the mathematical model that minimizes the number of fire-ups of the physical servers is given by

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

This model not only features the set of constraints (8) in addition to (1) - (4), but also the capacity constraint (6) is imposed for both the arrival and departure times of the VM requests.

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 counter examples below attest to the need for an objective function incorporating both criteria.

Example 2.1. *Figure 1 illustrates the first counter example, where the optimal solution of model (1) - (4) is not optimal for model (5) - (9). 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

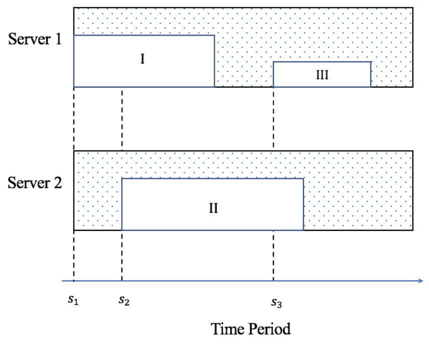


Figure 1: An example demonstrating that the optimal solution of (1) - (4) is not optimal for (5) - (9)

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) - (9). 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 counter example, where the optimal solution of model (5) - (9) 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.

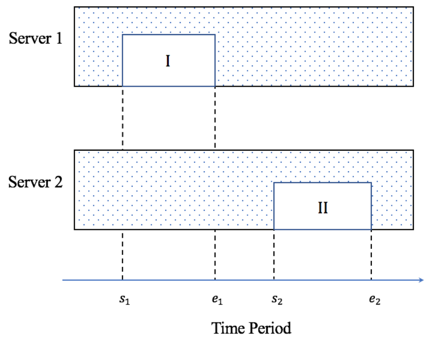


Figure 2: An example demonstrating that the optimal solution of (5) - (9) is not optimal for (1) - (4)

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. The multi-objective model that minimizes the number of used physical servers and the number of fire-ups is given as

$$
\begin{aligned}
(\mathbf{M1})\quad \text{minimize}\quad & \gamma \sum_{l\in\tau}\sum_{k\in K} w_l^k + \sum_{k\in K} z_k && && (10)\\
\text{subject to}\quad & y_l^k \le \sum_{i\in I} a_{il} c_i x_{ik} \le y_l^k C, && k\in K; l\in\tau, && (11)\\
& \sum_{k\in K} x_{ik} = 1, && i\in I, && (12)\\
& y_l^k - y_{l-1}^k \le w_l^k, && k\in K; l\in\tau, && (13)\\
& y_l^k \le z_k, && k\in K; l\in\tau, && (14)\\
& y_l^k \ge x_{ik}, && i\in I; k\in K, l\in\tau, t_l = s_i, && (15)\\
& x_{ik}\in\{0,1\}, && i\in I; k\in K, && (16)\\
& y_l^k\in\{0,1\}, w_l^k \ge 0, && i\in I; k\in K; l\in\tau, && (17)
\end{aligned}
$$

where $\gamma > 0$ is a scaling parameter. Constraint set (11) ensures that the total load on a physical server does not exceed its capacity and the physical server k is marked as idle (switched-off) if it is not in use at time t_l, $l \in \tau$. Constraints (12) guarantee that each VM request is assigned to a physical server. Constraint sets (13) and (14) are defined to keep track of the fire-ups in each physical server and the number of physical servers scheduled during the planning period, respectively. Similarly, constraints (15) ensure that the physical server k is marked as scheduled if it is in use at time t_l, $l \in \tau$. Constraints (15) are redundant due to (11), however, they tighten the LP relaxation of the model.

Next, we analyze the relationship between two objective functions in the M1 (M1) and show that 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 number of active servers obtained by solving* (10)-(17) *is equal to the optimal number of servers obtained by solving* (1)-(4).

Proof. Let Z^* and W^* be the optimal number of servers and optimal number of fire-ups obtained by solving the problem (10) - (17) for $\gamma = 1$, respectively. We first show for $\gamma = 1$ that in all feasible solutions of problem (10) - (17), the number of servers is greater than or equal to Z^*. This shall follow from contradiction. Suppose that $Z^* = 2$ and there is a feasible solution with one server and $\hat{W}$ fire-ups. We denote the VM requests assigned to servers 1 and 2 in the optimal schedule by the sets I_1 and I_2, respectively. If the optimal number of fire-ups on these servers are given by W_1^* and W_2^*, then, the total fire-ups is equal to $W^* = W_1^* + W_2^*$. When we move the assigned VM requests on server 2 to server 1, we can obtain the feasible solution with one server and $\hat{W}$ fire-ups. Next, we examine the relation between $\hat{W}$ and W_1^*. We have one of the following two cases on server 1.

1. $\hat{W} \leq W_1^*$: In this case, the objective value of the feasible solution is $1 + \hat{W}$, which is less than the optimal objective value, $2 + W_1^* + W_2^*$, and this contradicts the optimality of $Z^* = 2$ and $W^* = W_1^* + W_2^*$.
2. $\hat{W} > W_1^*$: Consider the optimal VM assignments on servers 1 and 2. Some of the available time slots in server 1 are already occupied by the VM requests in set I_1. When we move the VM requests in set I_2 to server 1, the number of fire-ups on server 1 is increased by $\hat{W} - W_1^*$. The increase in fire-ups cannot be more than the number of fire-ups on server 2 which is W_2^*. Thus, the objective value of the feasible solution is at most $1+\hat{W} = 1+W_1^*+W_2^*$, which is less than $2 + W_1^* + W_2^*$. This contradicts the optimality of solution $Z^* = 2$ and $W^* = W_1^* + W_2^*$.

We have established that we cannot have a feasible solution with one server when the optimal number of servers is two. The demonstration for $Z^* > 2$ can be constructed similarly. Assume that each server has infinite capacity and, hence, we can assign all m VM requests to one server. Next, consider the case where servers have capacity limitation and we find the optimal assignment solution with two servers. Since we allocate the m VM requests to two servers, the total fire-ups cannot be less than the one in the case of infinite capacity. Now assume that the capacities of the servers are further decreased and the optimal assignment is now given by three servers. Since we allocate the same VM requests to three servers, the total fire-ups becomes at least the number of fire-ups in the case of the two server solution. To sum up, if Z^* is the optimal number of active servers to schedule m VM requests, we cannot have a feasible solution with fewer number of servers. Otherwise, this feasible solution would have fewer number of servers and fire-ups. This contradicts the optimality of the solution with Z^* servers. We have just shown that the optimal number of scheduled physical servers obtained by solving problems (10) - (17) and (1)- (4) are the same for $\gamma = 1$. Moreover, the number of fire-ups increases, if we schedule the same number of VM requests to more physical servers. Therefore, for any $\gamma > 0$, solving model (10) - (17) minimizes the number of scheduled servers and the optimal number is the same as that obtained by solving (1)-(4). $\square$

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.

3. Bounding Methods

Being an extension of the VMP problem, problem (M1) is also NP-hard, which makes it important to find efficient solution strategies for large-scale problems. By analyzing the structural properties of the problem, we propose upper and lower bounding methods. In the following subsections, we first explain the proposed methods that provide an upper bound on the VMP problem. Then, we introduce an exact column generation algorithm, which gives a lower bound.

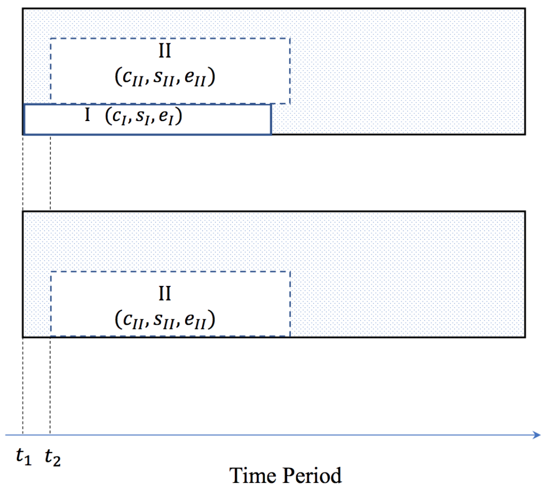
3.1. Upper Bounds

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. We also propose an iterative heuristic method for the VMP problem. 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.

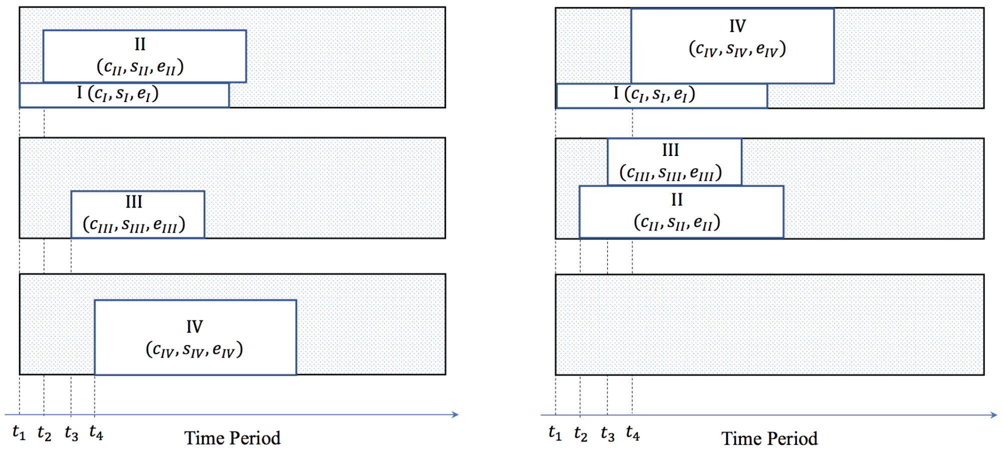
Constructive Look-ahead Heuristic (CLH)

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 time t_{l+1} and t_{l+2}. This concept is illustrated in Figure 3, where 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 the possible positions for the arriving request. As depicted in Figure 3(a), there are two possible positions to place VM request II; either in server 1 or in server 2. By considering the VM requests arriving at time period t_3 and t_4, we decide the best position which minimizes the total number of used servers and the total number of fire-ups. As illustrated in Figure 3(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. 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.

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 3: An illustration of VM assignment process in CLH

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 4-13). One of the eligible VM requests is selected and 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 the model M1. The current solution is updated with the new schedule and the process is repeated until the given stopping criterion is met. The proposed algorithm is terminated if the solution is not improved withing specified iterations. The overall structure of the algorithm is displayed in Algorithm 1.

The key component of this method is to generate the 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 15). 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 10-11). This step helps us to quickly eliminate some of the unavoidable fire-ups.

3.2. Lower Bound

To facilitate the demonstration and pave our way to the presentation of our proposed lower bounding method, 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 the total load on k is equal to the size of the request, c_i; *i.e.*, it is the only VM on server k at any given time. This observation obviates the need for $y-$variables, which are used along with the set of constraints (11) to indicate whether a server is used. In order to translate the load on a server into a constraint *without* using the time index

Algorithm 1: Recovery Algorithm

```
1:  Obtain initial solution $\hat{x} = \{\hat{x}_{ik} : i \in I, k \in K\}$ from CLH
2:  Initialize tabu list, iter = 0
3:  while stopping criterion not met do
4:      Initialize eligible VM set, $\xi \leftarrow \emptyset$
5:      Define fire-up set $\mathcal{S} = \{i \in I : w_i^k = 1, k \in K\}$
6:      for $i \in \mathcal{S}$ do
7:          Construct adjacency set $\mathcal{A}_i = \{j \in I : s_i \leq s_j \leq e_i \vee s_i \leq e_j \leq e_i\}$
8:          if iter = 0 then
9:              Compute the number of start-ups, $W_i = \sum_{k \in K} \sum_{j \in \mathcal{A}_i} w_j^k$
10:     if iter = 0 then
11:         $\xi \leftarrow \xi \cup i^*$ where $i^* = \arg\max_{i \in \mathcal{S}} \{W_i\}$
12:     else
13:         $\xi \leftarrow \xi \cup \mathcal{S}$
14:     Randomly select a VM $j$ which is not tabu from set $\xi$
15:     Add $j$ to tabu list; update tabu list
16:     Randomly select a subset of first degree neighbours $\mathcal{R}_1 \subset \mathcal{A}_j$
17:     Find the time interval $(s, e)$ for set $\mathcal{R}_1$
18:     Randomly select a subset of second degree neighbours
         $\mathcal{R}_2 \subset \{i \in I \setminus \mathcal{A}_j : s \leq s_i \leq e \vee s \leq e_i \leq e)$
19:     Set $x_{ik} = 1$ for $i \in I \setminus (\mathcal{R}_1 \cup \mathcal{R}_2)$ in M1
20:     Solve model M1
21:     iter = iter +1
22: return solution $(\hat{x}, \hat{y}, \hat{w})$
```

$l \in \tau$, 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 $\delta_i = \{j < i : e_j > s_i\}$ and $\delta_i^+ = \{j < i : e_j \geq s_i\}$. 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 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 can be written as

$$\textbf{(M2)} \quad \text{minimize} \quad \sum_{i \in I} \sum_{k \in K} w_i^k + \sum_{k \in K} z_k \tag{18}$$

$$\text{subject to} \quad \sum_{j \in \delta_i} c_j x_{jk} + c_i x_{ik} \leq C z_k, \qquad k \in K, i \in I, \tag{19}$$

$$x_{ik} \leq z_k, \qquad k \in K; i \in I, \tag{20}$$

$$\sum_{k \in K} x_{ik} = 1, \qquad i \in I, \tag{21}$$

$$\sum_{j \in \delta_i^+} c_j x_{jk} - x_{ik} + w_i^k \geq 0, \qquad k \in K; i \in I, \tag{22}$$

$$x_{ik}, z_k \in \{0,1\}, w_i^k \geq 0 \qquad i \in I; k \in K, \tag{23}$$

where (21) 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 (19) 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. We modify the index set of $w-$variables as w_k^i, $i \in I$ and $k \in K$, which indicates whether server k is switched on at s_i. This variable takes value one only if at s_i of a request i, which is assigned to server k, no request is active on this server, i.e. $\delta_i^+ = \emptyset$, as imposed in constraint set (22). 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 $|\tau| = 2|I|$, the capacity and fire-up controls are both handled at the start time of the requests whose cardinality is $|I|$. Comparing these two models M1 and M2, we conclude that in the latter, the number of variables shrinks by $3|I||K|$ due to the decrease in the size of $w-$variables and the elimination of $y-$variables. Also, the number of constraints reduces by $3|I| + |I||K|$, where $3|I|$ is due to the disparity between the index sets in (11), (13) and (14), namely τ and I, and $|I||K|$ is attributed to constraints (15) absent in M2. 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 3.1. *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 (19) due to (11) where $\{j \in I : a_{il} = 1\} = \delta_i \cup \{i\}$ for $t_l = s_i$ and (14). Moreover, (20) is implied by (14) and (15). Feasibility of (21) is trivial as it is equivalent to (12). To show feasibility of (22), 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 $l \in \tau$:

$$\hat{y}_l^k = \max\left\{\max_{i:a_{il}=1}\{\hat{x}_{ik}\}, \frac{\sum_{i\in I} a_{il}c_i\hat{x}_{ik}}{C}\right\}, \tag{24}$$

$$\hat{w}_l^k = \hat{y}_l^k - \hat{y}_{l-1}^k, \tag{25}$$

where the former is due to (11) and (15) and the latter follows from (13). Substituting the terms, we can rewrite (22) for $i \in I$, $k \in K$ and $t_l = s_i$ as

$$w_{ik} = \hat{y}_l^k - \hat{y}_{l-1}^k \geq \hat{x}_{ik} - \sum_{j\in\delta_i^+} c_j\hat{x}_{jk}. \tag{26}$$

The left-hand-side of the inequality is always non-negative, i.e. $\hat{y}_l^k - \hat{y}_{l-1}^k \geq 0$, since t_l corresponds to the start time s_i of request i in M2, which increases the load on k at t_l. If $\hat{x}_{ik} = 0$, this constraint is always satisfied as $\sum_{j\in\delta_i^+} c_j\hat{x}_{jk} \geq 0$. Otherwise, we arrange (26) as

$$\hat{y}_l^k - \hat{x}_{ik} \geq \hat{y}_{l-1}^k - \sum_{j\in\delta_i^+} c_j\hat{x}_{jk}, \tag{27}$$

where $\sum_{j\in\delta_i^+} c_j\hat{x}_{jk} = \sum_{j\in I} a_{jl-1}c_j\hat{x}_{jk}$ 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 (24), 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 (27) is satisfied using the following two inequalities:

$$\hat{y}_l^k - \hat{x}_{ik} \geq \max_{j:a_{jl-1}=1}\{\hat{x}_{jk}\} - \sum_{j\in\delta_i^+} c_j\hat{x}_{jk}, \tag{28}$$

$$\hat{y}_l^k - \hat{x}_{ik} \geq \hat{y}_{l-1}^k - C\hat{y}_{l-1}^k. \tag{29}$$

In both inequalities, the left-hand-side is non-negative due to (15). In (28), the right-hand-side is non-positive since $c_j \geq 1$, $j \in I$ and $\arg\max_{j:a_{jl-1}=1}\{\hat{x}_{jk}\} \in \delta_i^+$, while that in (29) is non-positive as $\hat{y}_{l-1}^k(1-C) \leq 0$ for $C \geq 1$, $k \in K$.

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 counter example with $|K| = 2$, $|I| = 3$, $C = 1$, $k \in K$ and $c_i = 1$, $i \in I$, for which $\hat{x}_{ik} = 1/2$ for all I and K. The solution is demonstrated in Figure 4. These $x-$values induce $\hat{z}_1 = 1$ and $\hat{z}_2 = 1$ due to constraints (19)-(20), 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 (22), the values of $w-$variables can then be found by $\hat{w}_{ik} = \max\left\{0, \hat{x}_{ik} - \sum_{j\in\delta_i^+} c_j\hat{x}_{jk}\right\}$, which yields $\hat{w}_1^1 = \hat{w}_1^2 = 1/2$ and $\hat{w}_i^k = 0$ for $i \in I\backslash\{1\}$ and $k \in K$. The corresponding objective function value is 3. This solution can be translated into the solution of M1 for $x-$variables directly, while $w-$variables can be projected to the time index set $l \in \tau$ only for the start times of requests $i \in I$ as $\hat{w}_l^k = 1/2$ for $t_l = s_1 = 0$ and $k \in K$, and $\hat{w}_l^k = 0$ for $t_l = s_2 = 1$, $t_l = s_3 = 2$ and $k \in K$. The values of the $y-$variables in M1 can be deduced using (24) such that $\hat{y}_l^1 = 1/2$ for $t_l = 0$ and $\hat{y}_l^1 = 1$ for $t_l = 1$. From (25), for $t_l = 1$, $\hat{w}_l^1 = \hat{y}_l^1 - \hat{y}_{l-1}^1 = 1 - 1/2 = 1/2$, which is not equal to its value stipulated by the solution of M2. Therefore, there is a feasible LP solution of M2 that is not feasible for M1.

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

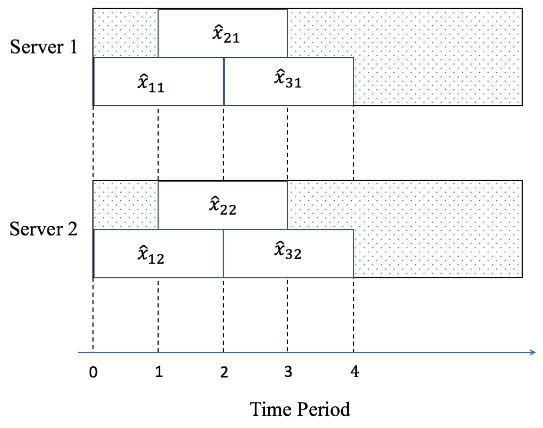


Figure 4: An example demonstrating that a feasible solution of M2 is not feasible for M1

□

Constraint sets (19) and (22) exhibit a block-angular structure, which is decomposable for each server k. Each solution satisfying (19) and (22) for a server k is a feasible schedule, which satisfies the capacity constraints and those constraints facilitating the calculation of fire-ups through w_i^k for each i. 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:

$$\begin{aligned}
(\mathbf{MP})\quad \text{minimize} \quad & \sum_{s\in S} r_s\lambda_s && && (30)\\
\text{subject to} \quad & \sum_{s\in S} a_{is}\lambda_s = 1, && i\in I, && (31)\\
& \sum_{s\in S} \lambda_s \le |K|, && && (32)\\
& \lambda_s \in \{0,1\}, && s\in S, && (33)
\end{aligned}$$

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 number of fire-ups, and a_{is} is a binary parameter taking value one, only if request i is covered in schedule s. While constraints (32) ensure that each request exists in exactly one of the selected schedules, constraint set (32) limits the number of selected schedules to the number of available servers.

MP is a set partitioning problem with an extra constraint (32), which contains many 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 (33). The solution of RMP offers the dual information associated with the constraint sets (31) and (32), 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. 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 only propose a column generation algorithm to find a lower bound, which is used to evaluate the performance of our proposed heuristic. In the next section, we show through computational study that the proposed algorithm produces tight lower bounds for instances up to 200 requests.

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

$$\min_{s \in S} \{r_s - \sum_{i \in I} u_i a_{is} - v\}, \tag{34}$$

where u_i, $i \in I$ are the dual variables associated with the constraints (31), v is the dual variable associated with the constraint (32) 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 $w_i = 1$. We can write the PSP as

$$
\begin{aligned}
(\mathbf{PSP}) \quad \text{minimize} \quad & \sum_{i \in I} (w_i - u_i x_i) + 1 - v && && (35)\\
\text{subject to} \quad & \sum_{j \in \delta_i} c_j x_j + c_i x_i \le C, && i \in I, && (36)\\
& \sum_{j \in \delta_i^+} c_j x_j - x_i + w_i \ge 0, && i \in I, && (37)\\
& x_i \in \{0,1\}, && i \in I, && (38)\\
& w_i \in \{0,1\}, && i \in I, && (39)
\end{aligned}
$$

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 and o^d 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 (36) 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 ($R_A^p = \delta_i \cup \{i\}$), respectively, R_L^p represents the load of active requests of partial path p, hence can be defined as $R_L^p = \sum_{j \in \delta_i \cup i} c_j x_j$. 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 (35). This algorithm runs in $|V|-1$ steps, one for each node except o^d, which are executed in the topological order of V.

The algorithm starts with a null label at o^s whose resources are initialized as zero and empty sets except that $R^p_{\mathcal{C}} = 1-v$. At step i of the algorithm, the node in the i^{th} position is treated, which involves extending all the partial paths, and their related labels, on that node to all the nodes in the j^{th} position where $j > i$. The extension of a label X^i_p associated with path p on node i to j to form a new label X^j_q related with path q involves updating the values of the resource with the addition of the new node j, which is trivial for $R^q_P = R^p_P \cup \{q\}$. We first check whether $R^p_E < s_j$ holds, in which case $R^q_E = e_j$, $R^q_A = \{q\}$, $R^q_L = 0$ and $R^q_{\mathcal{C}} = R^p_{\mathcal{C}} + d_{ij} + 1$ due to the fire-up of the idle server with the start of j. Otherwise, $R^q_{\mathcal{C}} = R^p_{\mathcal{C}} + d_{ij}$, $R^q_A = R^p_A \backslash \{k \in R^p_A : e_k < s_j\} \cup \{j\}$, which induces recalculation of R^q_L, and $R^q_E = e_j$ if $e_j > R^p_E$, and $R^q_E = R^p_E$, otherwise. The extension of the label is feasibly effectuated only if $R^q_L \leq C$. 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 $R^q_{\mathcal{C}} < 0$ 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 3.2. *Label X^i_p on node i dominates another label X^i_q on this node, if $R^p_{\mathcal{C}} \leq R^q_{\mathcal{C}}$, $R^p_E = R^q_E$ and $R^p_A \subseteq R^q_A$.*

Proof. To prove this proposition, we show that any feasible extension of X^i_q from node i to o^d is also feasible for X^i_p, and in all these extensions, X^i_q never attains smaller reduced cost than X^i_p. First, due to $R^p_A \subseteq R^q_A$ and $R^p_E = R^q_E$ as given in the proposition, $R^p_L \leq R^q_L$ also holds, hence, X^i_p can visit more nodes $j > i$, which satisfy $s_j \leq R^p_E = R^q_E$, than X^i_q without incurring a fire-up cost. For the extensions starting with a node $j > i$, which satisfy $s_j > R^p_E = R^q_E$, these two labels can reach o^d with the same additional reduced cost. Therefore, in neither cases, the extension of X^i_q can attain a smaller reduced cost than that of X^i_p since $R^p_{\mathcal{C}} \leq R^q_{\mathcal{C}}$. $\square$

Implementation Details

In order to accelerate the solution of the labeling algorithm, we aim to eliminate the unpromising partial path as early as possible in the iterations. A partial paths 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. Note that the minimum cost reverse path from o^d to a node i, denoted by θ_i, does not include those nodes j for which $u_j < 0$. Therefore, if extending a label to node i that forms partial path p renders $R_C^p + \theta_i \geq 0$, then the extension of this partial path to i 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 3.3. *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 $R_C^q = R_C^p + d_{ij} = R_C^p - u_j \geq R_C^p$, 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 the 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.

4. 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 Xeon CPU E5-1505M (2.8 GHz) 32 GB Ram. CPLEX 12.8 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 15 iterations.

4.1. 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 ($\bar{s}$), the duration of the VM requests (d_i) and the requested capacities (c_i). In our numerical experiments, we consider various problem sizes with $n = \{50, 100, 150, 200, 500, 1000\}$. VM requests arrive over time period $\bar{s}$ such that $\max_{i \in I} s_i \leq \bar{s}$. The length of the VM arrival period 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. 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. The requested capacity of VM request i is computed by using $c_i = c_{min} + U(0, Q)$ where U is a function that generates a random integer in the given interval, Q is the maximum requested capacity and c_{min} is the minimum capacity demand. In our experiments, we set $c_{min} = 25$, and test the models for varying capacity requirements $Q \in \{50, 75\}$. 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) LBM: Lower bounding method; (ii) M1-OPT: Optimal solution of mixed integer programming model $M1$; (iii) RMAT: Recovery heuristic; (iv) CLH: Constructive look-ahead heuristic; (v) BF: Best Fit heuristic; (vi) SMM: Server minimization model.

4.2. Numerical Results

In this section, we present and discuss our results from the experiments carried out. Table 1 reports the results of LBM, RMAT, CLH and the solution of the exact model M1 denoted as M1-OPT for moderate size problems with $n = \{50, 100, 150, 200\}$. The individual results for each test instance are given by Tables 4 - 7 in Appendix. The first four columns in Table 1 show the characteristics of the test instances. The next three columns present the comparative measures for LBM, namely the average CPU time (in seconds) to find the optimal LP relaxation of MP, the number of instances for which LBM solves the LP relaxation of MP within the time limit, the average percentage gap with M1-OPT. 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. The last five 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, average percentage gaps with M1-OPT, CLH and LBM. A noteworthy observation is that the optimality of RMAT solution can be identified by comparing it with the M1-OPT and LBM solutions reported in Tables 4 - 7 in Appendix. For instance, both LBM and RMAT obtain the objective value of 51 in the first instance of test $(150, d_L, c_H)$ with $\bar{s} = 150$ in Table 6, while M1-OPT could not find the optimal solution within 30 minutes and terminates with the objective value of 53.

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 2%. On the other hand, when we compare the CLH and RMAT, we observe that RMAT improves the VM schedule obtained by CLH significantly. The percentage gap is more striking when the number of VM requests is high.

When we look into the optimality gap of LBM, we observe that it 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 5% in some cases, the optimal objective values of LBM and M1-OPT is actually very close. When we examine the individual test results given by Tables 4 - 7, we observe that the difference between the optimal objective values of these two methods is less than two.

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

Instances				Lower Bound (LBM)			M1-OPT		Recovery Mat (RMAT)				
n	$\bar{s}$	d_i	c_i	Time	Succ	Gap	Time	Opt	Time	Opt	M1-OPT Gap	CLH Gap	LBM Gap
50	50	d_S	c_L	19.8	5	4.21	42.0	5	18.7	5	0.0	10.09	4.21
			c_H	2.1	5	0.0	22.3	5	23.2	5	0.0	19.28	0.0
		d_L	c_L	15.4	5	2.40	487.2	3	69.2	3	0.0	6.19	3.90
			c_H	0.9	5	0.73	69.6	5	53.4	5	0.0	6.70	0.73
	60	d_S	c_L	14.7	5	4.05	35.1	5	3.9	5	0.0	14.23	4.05
			c_H	4.0	5	1.98	43.3	5	6.9	5	0.0	15.60	1.98
		d_L	c_L	15.2	5	2.73	294.8	4	116.8	4	0.0	7.39	3.59
			c_H	0.4	5	0.54	83.1	5	45.5	5	0.0	7.44	0.54
100	100	d_S	c_L	430.6	5	5.60	384.2	5	31.8	5	0.0	23.20	5.60
			c_H	68.8	5	0.63	297.8	5	72.9	4	0.61	24.17	1.23
		d_L	c_L	537.0	5	-	1800	-	63.4	-	-	10.36	6.32
			c_H	48.4	5	0.0	1136.3	1	195.8	-	1.72	14.33	3.37
	120	d_S	c_L	611.8	5	5.36	341.5	5	13.5	5	0.0	27.25	5.36
			c_H	123.4	5	1.61	578.3	5	70.3	5	0.0	29.53	1.61
		d_L	c_L	269.6	5	-	1800	-	16.9	-	-	8.98	7.52
			c_H	40.4	5	0.0	1800	-	108.04	1	-	19.13	3.37
150	150	d_S	c_L	1800	0	-	305.6	5	35.9	4	0.91	33.91	-
			c_H	801.6	5	0.0	1800	-	210.4	1	-	30.88	3.83
		d_L	c_L	955.3	4	0.66	1800	-	40.4	1	-	10.07	5.32
			c_H	201.4	5	-	1800	-	121.4	-	-	21.6	4.00
	180	d_S	c_L	130.0	1	0.0	399.2	5	34.81	5	0.0	38.10	0.0
			c_H	1453.7	3	-	1800	-	62.5	-	-	31.71	4.04
		d_L	c_L	1800	0	-	1800	-	12.8	-	-	10.70	-
			c_H	359.6	5	0.0	1800	-	115.1	1	-	24.15	4.53
200	200	d_S	c_L	1800	0	-	1800	-	31.9	-	-	36.10	-
			c_H	1800	0	-	1800	-	146.9	-	-	41.18	-
		d_L	c_L	1800	0	-	1800	-	42.2	-	-	13.89	-
			c_H	851	2	-	1800	-	282.4	-	-	26.07	5.96
	240	d_S	c_L	1800	0	-	451.6	5	63.9	5	0.0	45.59	-
			c_H	1800	0	-	1800	-	314.6	-	-	35.07	-
		d_L	c_L	1800	0	-	1800	-	69.3	-	-	21.81	-
			c_H	1603.3	3	-	1800	-	282.1	-	-	30.02	7.15

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 LBM, 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.

Next, we report our results for larger problems with $n = \{500, 1000\}$. Since LBM and M1-OPT cannot be solved to optimality for these instances, we compare the performances of RMAT, CLH and BF. 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 last experiment, we compare the performances of M1-OPT, RMAT and the solution of the model (1)-(4) denoted as SMM. Table 3 reports the average CPU times, average number of physical servers and the average number of fire-ups obtained by these solution methods. The first four columns in Table 3 show the characteristics of the test instances. The next column shows the number of test instances solved out of five within the specified time limits by these methods. The next nine 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. Note that, in Table 3, we only report the test results where both SMM and M1-OPT are solved to optimality. As Table 3 depicts, the optimal number of servers is equal for SMM and M1-OPT in all instances, which confirms Proposition 2.1. 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 3).

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

Instances				RMAT			CLH			BF		
n	$\bar{s}$	d_i	c_i	Time	z	w	Time	z	w	Time	z	w
500	500	d_S	c_L	45.72	12.8	16.0	1.35	13.2	54.6	0.09	13.2	61.0
			c_H	215.09	18.8	49.0	1.29	18.8	115.0	0.06	19.4	131.6
		d_L	c_L	69.32	22.0	23.4	1.47	22.2	46.0	0.06	22.4	50.8
			c_H	408.57	29.8	42.8	1.57	30.2	102.8	0.07	30.4	113.0
	600	d_S	c_L	52.76	11.4	16.8	1.47	12.0	53.2	0.07	11.8	62.0
			c_H	145.82	15.6	55.2	1.22	15.6	123.6	0.06	16.2	140.2
		d_L	c_L	43.37	18.8	19.6	1.61	19.0	46.8	0.09	19.6	51.8
			c_H	274.46	25.8	47.0	1.64	26.0	104.2	0.07	26.6	112.0
1000	1000	d_S	c_L	79.33	13.2	24.0	10.60	13.4	101.8	0.15	13.6	118.0
			c_H	363.32	21.2	96.0	6.11	21.8	237.8	0.16	21.2	265.2
		d_L	c_L	117.59	21.4	23.4	13.46	21.4	77.0	0.28	22.0	80.4
			c_H	765.37	33.2	86.0	8.93	33.2	205.0	0.24	34.2	221.0
	1200	d_S	c_L	69.62	11.8	24.8	7.36	12.0	103.2	0.19	12.0	121.2
			c_H	554.60	17.8	102.6	5.57	17.8	244.8	0.18	18.2	284.8
		d_L	c_L	120.58	19.2	21.6	8.69	19.2	84.0	0.26	19.6	87.2
			c_H	599.07	27.2	76.8	8.15	27.2	204.6	0.27	27.6	227.2

5. 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

Table 3: Average server and fire-ups

Instances				Num of	SMM			M1-OPT			RMAT		
n	$\bar{s}$	d_i	c_i	Tests	Time	z	w	Time	z	w	Time	z	w
50	50	d_S	c_L	5	5.5	9.8	11.2	42.0	9.8	9.8	18.7	9.8	9.8
50	50	d_S	c_H	5	5.8	12.8	18.8	22.3	12.8	13.0	23.2	12.8	13.0
50	50	d_L	c_L	3	97.9	15.0	15.3	487.2	15.0	15.0	69.2	15.0	15.0
50	50	d_L	c_H	5	25.6	23.0	26.4	69.6	23.0	23.0	53.4	23.0	23.0
50	60	d_S	c_L	5	10.7	8.4	10.0	35.1	8.4	8.4	3.9	8.4	8.4
50	60	d_S	c_H	5	8.6	11.8	19.4	43.3	11.8	12.2	6.9	11.8	12.2
50	60	d_L	c_L	3	167.1	14.3	15.0	141.5	14.3	14.3	36.7	14.3	14.3
50	60	d_L	c_H	5	30.0	20.8	23.2	83.1	20.8	20.8	45.5	20.8	20.8
100	100	d_S	c_L	3	19.1	11.0	15.7	134.4	11.0	11.0	16.8	11.0	11.0
100	100	d_S	c_H	5	137.8	16.8	42.4	297.8	16.8	18.0	72.9	17.0	18.0
100	100	d_L	c_H	1	152.7	28.0	53.0	1136.3	28.0	29.0	195.8	28.0	30.0
100	120	d_S	c_L	3	23.7	9.7	17.0	365.8	9.7	9.7	16.6	9.7	9.7
100	120	d_S	c_H	5	134.8	13.6	38.8	578.3	13.6	17.0	70.3	13.6	17.0
150	150	d_S	c_L	4	78.7	10.8	16.5	298.9	10.8	10.8	23.7	10.8	10.8
150	180	d_S	c_L	5	172.8	9.8	23.4	399.2	9.8	9.8	34.8	9.8	9.8
200	240	d_S	c_L	5	170.9	10.6	39.4	451.6	10.6	11.0	63.9	10.6	11.0

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.

Appendix

Table 4: Computational results for each test instance ($n = 50$)

Instances	$\bar{s} = 50$				$\bar{s} = 60$			
(n, d_i, c_i)	LBM	M1-OPT	RMAT	CLH	LBM	M1-OPT	RMAT	CLH
(50, d_S, c_L)	17.2	18	18	20	15.5	16	16	19
	19.6	20	20	22	15.4	16	16	19
	17.0	18	18	20	16.5	18	18	19
	21.1	22	22	24	16.0	16	16	20
	19.2	20	20	23	17.3	18	18	21
(50, d_S, c_H)	24.0	24	24	32	23.0	24	24	27
	28.0	28	28	36	29.0	29	29	34
	27.0	27	27	33	22.1	23	23	27
	28.0	28	28	32	24.0	24	24	28
	22.0	22	22	27	19.7	20	20	26
(50, d_L, c_L)	31.5	32	32	34	27.2	28	28	30
	28.4	30†	30	32	31.0	32	32	36
	27.5	28	28	30	28.8	30	30	32
	28.9	30	30	32	28.1	30†	30	32
	30.0	32†	32	34	27.9	28	28	30
(50, d_L, c_H)	36.7	38	38	40	44.0	44	44	49
	44.0	44	44	46	37.0	38	38	40
	48.0	48	48	52	28.0	28	28	42
	54.0	54	54	59	48.0	48	48	52
	46.0	46	46	50	40.0	40	40	42

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

Table 5: Computational results for each test instance ($n = 100$)

Instances	$\bar{s} = 100$				$\bar{s} = 120$			
(n, d_i, c_i)	LBM	M1-OPT	RMAT	CLH	LBM	M1-OPT	RMAT	CLH
(100, d_S, c_L)	19.0	20	20	29	19.1	20	20	25
	21.7	22	22	31	20.5	22	22	28
	22.8	24	24	31	17.7	18	18	26
	22.3	24	24	32	19.2	20	20	31
	20.3	22	22	24	18.4	20	20	28
(100, d_S, c_H)	41.0	41	41	50	34.0	35	35	48
	35.0	35	35	46	28.0	28	28	41
	33.0	33	34	46	31.5	32	32	47
	32.0	33	33	41	26.5	27	27	39
	32.0	32	32	48	30.5	31	31	42
(100, d_L, c_L)	33.9	38†	36	40	28.2	30†	30	32
	35.7	38†	38	41	29.2	32†	32	35
	33.9	36†	36	39	27.7	30†	30	33
	33.7	37†	36	43	32.5	34†	34	39
	32.1	34†	34	38	31.2	34†	34	37
(100, d_L, c_H)	44.0	48†	46	56	40.3	46†	44	55
	57.0	57	58	70	52.0	53†	52	65
	48.0	50†	50	56	40.7	42†	42	55
	44.0	48†	46	54	47.0	49†	48	60
	55.0	56†	56	63	43.0	44†	44	50

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

Table 6: Computational results for each test instance ($n = 150$)

Instances	$\bar{s} = 150$				$\bar{s} = 180$			
(n, d_i, c_i)	LBM	M1-OPT	RMAT	CLH	LBM	M1-OPT	RMAT	CLH
(150, d_S, c_L)	-	20	20	34	20.0	20	20	33
	-	22	22	33	-	18	18	33
	-	20	20	31	-	18	18	28
	-	24	24	35	-	20	20	29
	-	22	23	32	-	22	22	36
(150, d_S, c_H)	40.0	$41^{\dagger}$	42	55	-	$42^{\dagger}$	42	52
	44.5	$51^{\dagger}$	46	65	25.0	$26^{\dagger}$	26	46
	41.0	$41^{\dagger}$	41	55	-	$37^{\dagger}$	36	54
	32.5	$36^{\dagger}$	35	56	37.0	$40^{\dagger}$	38	56
	34.0	$47^{\dagger}$	35	57	37.0	$39^{\dagger}$	39	56
(150, d_L, c_L)	37.0	$40^{\dagger}$	40	45	-	$34^{\dagger}$	34	39
	39.8	$42^{\dagger}$	42	46	-	$36^{\dagger}$	36	37
	39.3	$42^{\dagger}$	42	43	-	$40^{\dagger}$	34	39
	-	$42^{\dagger}$	40	44	-	$37^{\dagger}$	35	41
	37.8	$40^{\dagger}$	38	47	-	$34^{\dagger}$	34	38
(150, d_L, c_H)	47.3	$67^{\dagger}$	51	68	51.0	$53^{\dagger}$	51	72
	55.0	$95^{\dagger}$	57	67	42.8	$74^{\dagger}$	45	58
	56.0	$91^{\dagger}$	58	74	52.2	$92^{\dagger}$	55	77
	57.0	$91^{\dagger}$	58	72	52.0	$88^{\dagger}$	54	67
	48.5	$77^{\dagger}$	51	70	48.0	$91^{\dagger}$	52	66

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

Table 7: Computational results for each test instance ($n = 200$)

Instances	$\bar{s} = 200$				$\bar{s} = 240$			
(n, d_i, c_i)	LBM	M1-OPT	RMAT	CLH	LBM	M1-OPT	RMAT	CLH
(200, d_S, c_L)	-	22	22	35	-	22	22	38
	-	26	24	41	-	20	20	36
	-	29	28	41	-	22	22	42
	-	24	25	42	-	20	20	37
	-	24	26	37	-	24	24	46
(200, d_S, c_H)	-	73†	44	72	-	49†	42	65
	-	66†	43	71	-	51†	40	67
	-	37†	38	62	-	46†	42	66
	-	36†	30	54	-	55†	50	74
	-	50†	35	63	-	53†	47	68
(200, d_L, c_L)	-	52†	40	50	-	39†	36	47
	-	47†	40	47	-	40†	32	37
	-	43†	42	44	-	49†	36	45
	-	45†	38	43	-	38†	36	49
	-	49†	40	49	-	42†	32	43
(200, d_L, c_H)	-	96†	56	76	-	110†	53	81
	58.0	110†	59	80	42.0	96†	43	71
	49.0	78†	54	71	-	108†	57	73
	-	115†	57	84	50.0	105†	53	73
	-	116†	65	83	45.0	115†	52	71

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

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.

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.

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.

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.

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.

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

Jennings B, Stadler R. Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management 2015;23:567–619.

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.

Lee S, Panigrahy R, Prabhakaran V, Ramasubramanian V, Talwar K, Uyeada L, Wieder U. Validating heuristics for virtual machines consolidation. Technical Report; Microsoft Research Silican Valley, California; 2011.

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.

Puschel T, Schryen G, Hristova D, Neumann D. Revenue management for cloud computing providers: Decision models for service admission control under non-probabilistic uncertainty. European Journal of Operational Research 2015;244:637–47.

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.

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.