Hybrid Algorithms for Placement of Virtual Machines across Geo-Separated Data Centers

Fernando Stefanello[a,*], Vaneet Aggarwal[b], Luciana S. Buriol[a], Mauricio G.C. Resende[c,d]

[a] *Instituto de Informática, Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil.*
[b] *School of Industrial Engineering Purdue University, West Lafayette, IN 47907 USA.*
[c] *Amazon.com, Seattle, WA 98109 USA.*
[d] *Industrial and Systems Engineering, University of Washington, Seattle, WA 98105 USA.*

Abstract

Cloud computing has emerged as a new paradigm for hosting and supplying services over the Internet. This technology has brought many benefits, such as eliminating the need for maintaining expensive computing hardware. With an increasing demand for cloud computing, providing performance guarantees for applications that run over cloud become important. Applications can be abstracted into a set of virtual machines with certain guarantees depicting the quality of service of the application. In this paper, we consider the placement of these virtual machines across multiple data centers (VMPlacement), meeting the quality of service requirements while minimizing the bandwidth cost of the data centers. This problem is a generalization of the NP-hard Generalized Quadratic Assignment Problem (GQAP). In this paper, we present a Greedy Randomized Adaptive Search Procedure (GRASP) and a Biased Random-Key Genetic Algorithm (BRKGA), both hybridized with a Path-Relinking strategy and a local search based on Variable Neighborhood Descent (VND) for solving this problem. The hybrid heuristics are also tested on instances of the GQAP. We show that both algorithms are effective in quickly solving small and large instances of VMPlacement problem, especially when the path-relinking is used. For GQAP, the results outperform the previous state-of-the-art algorithms.

Keywords: Combinatorial optimization, Cloud computing, Biased Random-Key Genetic Algorithm, GRASP, Path-Relinking

1. Introduction

Virtualization of physical servers has gained prominence in enterprise data centers. This is because virtualization offers virtually unlimited resources without any upfront capital investment and a simple pay-as-you-go charging model. Long-term viability of virtualization

*Corresponding author.
Email addresses: fstefanello@inf.ufrgs.br (Fernando Stefanello), vaneet@purdue.edu (Vaneet Aggarwal), buriol@inf.ufrgs.br (Luciana S. Buriol), mgcr@uw.edu (Mauricio G.C. Resende)

depends, among other factors, on cost and performance. To attain performance guarantees, application providers can offer requirements for a number of virtual machines, bandwidth/latency requirements between virtual machines, and latency requirements between users of the service and virtual machines. Once a service provider can provide these performance guarantees, an optimized service can be offered to user applications. However, the service provider has to match the requirements of different applications to the placement of virtual machines with the limited bandwidth links between geographically separated data centers while minimizing its cost.

Unfortunately, today's public cloud platforms such as Amazon EC2[1] do not provide any performance guarantee, which in turn affects tenant cost. Specifically, the resource reservation model in today's clouds only provisions CPU and memory resources but ignores networking completely. Because of the largely oversubscribed nature of data center networks (e.g., Greenberg et al. (2009)), network bandwidth is a scarce resource shared across many tenants. In order to meet reliability and demand requirements, data centers have to be located all around the world. For instance, a teleconference call connects people from all over the world, and a data center within a reasonable distance to the end users is needed.

For distributed data centers, networking cost is one of the major costs. Given the limited bandwidth links between data centers, communication-intensive phases of applications collide and compete for the scarce network resources, which leads to unpredictable running times. The uncertainty in execution time further translates into unpredictable cost as tenants need to pay for the reserved virtual machines (VMs) for the entire duration of their jobs.

Placement of virtual machines within a data center has been widely explored (Guo et al., 2010; Piao and Yan, 2010; Ballani et al., 2011; Xie and Hu, 2012; Alicherry and Lakshman, 2012; Biran et al., 2012; Guo et al., 2017). These papers account for the networking needs in addition to the CPU and memory needs within a data center. For example, Guo et al. (2010) propose bandwidth reservation between every pair of VMs. Ballani et al. (2011) propose a simpler virtual cluster (VC) model where all virtual machines are connected to a virtual switch with links of bandwidth B. Xie and Hu (2012) extend these approaches to consider time-varying network requirement. Alicherry and Lakshman (2012) develop resource allocation algorithms for distributed cloud systems where the primary objective is to minimize the maximum latency in communication between the virtual machines allocated for a user request. However, in comparison with our work, all these papers account for a single data center or do not ensure minimum network requirements, or consider different allocation resource objectives. Instead, this paper deals with the placement of virtual machines across geo-separated data centers, ensuring network resource requirement guarantees on interconnections, and communication cost minimization. To the best of our knowledge, this problem with these requirements has not been accounted for in the prior work.

In this paper, we consider multiple data centers that are connected with limited bandwidth links. The latency between every pair of data centers is known. Each data center provides a finite number of resources to users, called virtual machine (VM). A virtual ma-

[1] http://aws.amazon.com/ec2/

chine is part of the data center resource requested by users to execute user-defined tasks that can exchange data with other tasks placed in others VMs. We assume that all VMs have identical data center resource consumption. In order to meet the application's quality of service guarantees, there is an imposed minimum bandwidth and maximum latency between each pair of virtual machines. We assume that there are multiple users who would use these services, and users are connected to some data center. In order to meet the overall application performance, there is an additional requirement of maximum latency between users and the virtual machines. Intuitively, if there is a set of VMs needed by a user and the set does not have any requirement with any other user or VM, it can be placed in a single data center. However, a VM interacts with multiple VMs which may be needed by other users, thus increasing the set of options for placement. There is a cost of transferring data between data centers and the placement minimizes this cost thus preferring placement of all VMs in a single data center which may not be feasible due to the quality of service requirements. Note that in this paper we consider a simplified version of a real problem. Some characteristics as virtual machine CPU and memory requirement, virtual machine relocation and many others are not considered here.

This problem has similarity with the problem of *Virtual Network Embedding* (VNE), since the set of virtual machines and the relation between then can be considered as the *Virtual Network* (VN), while the set of data centers and the links can be mapped as the *Substrate Network* (SN). Fischer et al. (2013) describe a general framework for VNE. The VMPlacement problem can be described using this framework, but the problem has some differences from other works since we consider a complete graph connecting the data center, a quadratic cost function for the communication cost, and latency constraints that usually are ignored from other works.

This problem is a generalization of the NP-hard Generalized Quadratic Placement Problem (GQAP) given in Lee and Ma (2004). Solving this problem optimally is possible only for very small instances, which may not represent the sizes found in real-world applications. Thus, we propose two heuristic approaches to solve the Virtual Machine Placement Problem (VMPlacement). We extend the BRKGA (Gonçalves and Resende, 2011) proposed in Stefanello et al. (2015a) by including new neighborhood structures in the local search and proposed a new GRASP algorithm, both coupled with a path-relinking (Glover, 1989) and an extensive local search procedure. We test the performance of both algorithms on problems from a dataset comprised of instances of sizes ranging from small to large. Both algorithms have similar performance, although a slight advantage for BRKGA was observed in small instances while GRASP has a slight advantage in larger instances. We show that the algorithms are able to quickly find feasible solutions and find high-quality final solutions for the VMPlacement problem, especially when the path-relinking procedure is used. Furthermore, the results obtained with the proposed algorithms outperforms the previous state-of-the-art results for GQAP.

This paper is an extension of Stefanello et al. (2015a) and Stefanello et al. (2015b) and has several additional contributions not found in the previous papers: i) We propose a path-relinking strategy as an intensification method, and a local search method based on a large exploration of the neighborhood; ii) We extend the BRKGA proposed in Stefanello et al. (2015a) to include new neighborhood structures in the local search, and propose a new GRASP algorithm; iii) We integrate a path-relinking strategy to BRKGA; iv) We provide experimental results for CPLEX, BRKGA and GRASP applied to a set of instances for VMPlacement and GQAP.

The remainder of the paper is organized as follows. In Section 2, we present mathematical models for the Virtual Machine Placement Problem in multiple data centers. Two heuristic approaches are described in Section 3. Computational results are presented in Section 4. Finally, conclusions are drawn in Section 5.

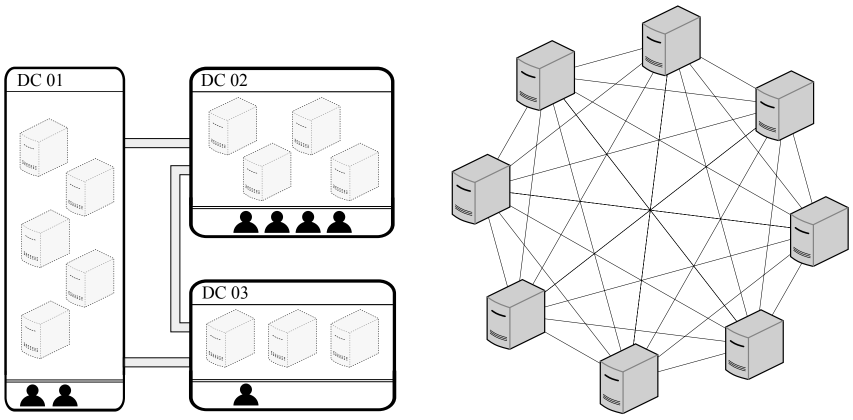
2. Virtual Machine Placement Problem

In the Virtual Machine Placement Problem (VMPlacement), the objective is to place a set K of virtual machines (VM) in a set N of data centers (DC) in order to minimize the communication cost among virtual machines over the network.

In this problem, each data center has a capacity a_i, which represents the number of virtual machines that can be placed in DC i. Also, between two data centers i and j, there is a bandwidth capacity (B_{ij}), a latency (L_{ij}), and a cost C_{ij} to transfer a data unit between the pair of data centers. In order to meet the reliability and demand requirements of the applications, certain bandwidth and latency requirements can be imposed on the different VMs that are placed on the data centers. Each pair of virtual machines v and w has a required bandwidth (b_{vw}) whose sum overall VMs placed between DCs i and j cannot exceed B_{ij}. Furthermore, there is a required latency (l_{vw}), such that VMs v and w cannot be placed in data centers i and j if the required latency is greater than the respective data center latency. Finally, there is a set U of users who access the system. Each user u is located at a data center $d(u)$ and has a maximum latency requirement t_{uv} for each VM v.

Figure 1 from Stefanello et al. (2015a) shows a representation of the input data components: the data centers (Figure 1a), and the virtual machines (Figure 1b). The first component is composed of three data centers (rounded rectangles). Each data center has a number of users and a capacity (represented as a number of spots where VMs can be placed). The connection between each pair of DCs represents the bandwidth capacity, latency, and cost. The second component is composed of eight virtual machines, where each link represents the required bandwidth and the required latency.

In the next subsections, we present three mathematical formulations for the VMPlacement problem. We first present a quadratic formulation, followed by two mixed integer linear formulations. Since VMPlacement is a generalization of the NP-hard Generalized Quadratic Assignment Problem, we extend the linear mathematical models proposed for the GQAP in Lee and Ma (2004) to VMPlacement. The models in Lee and Ma (2004) were also extended from the mixed integer linear programming formulation from Kaufman and Broeckx (1978) and Frieze and Yadegar (1983) to the Quadratic Assignment Problem, all of them based on the formulation for the QAP described in Koopmans and Beckmann (1957).



(a) Data centers. (b) Virtual machines.

Figure 1: Input data representation.

2.1. Quadratic mathematical model

A natural formulation for VMPlacement is based on a quadratic formulation as a generalization of GQAP. In what follows we summarize the parameters and present a quadratic mathematical model for VMPlacement (QMVMP) introduced in Stefanello et al. (2015a).

Parameters:

N:	set of data centers;
K:	set of virtual machines;
U:	set of users;
a_i:	capacity in number of VMs that DC i can host;
B_{ij}:	bandwidth between DCs i and j;
L_{ij}:	latency between DCs i and j;
C_{ij}:	cost of transferring a data unit between DCs i and j;
b_{vw}:	required bandwidth between VMs v and w;
l_{vw}:	required latency between VMs v and w;
$d(u)$:	DC that hosts user u;
t_{vu}:	required latency between user u and VM v;
c_{iv}:	cost of placing a VM v in a DC i.

Equations (1)-(7) present the quadratic model for the VMPlacement (QMVMP), where the binary decision variable $x_{iv} = 1$ if VM v is located in DC i, and $x_{iv} = 0$ otherwise.

$$\min \sum_{i\in N}\sum_{v\in K} c_{iv}x_{iv} + \sum_{i\in N}\sum_{j\in N}\sum_{v\in K}\sum_{w\in K} x_{iv}x_{jw}C_{ij}b_{vw} \tag{1}$$

subject to:

$$\sum_{v\in K} x_{iv} \le a_i \qquad \forall\, i \in N, \tag{2}$$

$$\sum_{i\in N} x_{iv} = 1 \qquad \forall\, v \in K, \tag{3}$$

$$\sum_{v\in K}\sum_{w\in K} x_{iv}x_{jw}b_{vw} \le B_{ij} \qquad \forall\, i,j \in N, \tag{4}$$

$$\sum_{i\in N}\sum_{j\in N} x_{iv}x_{jw}L_{ij} \le l_{vw} \qquad \forall\, v,w \in K, \tag{5}$$

$$\sum_{i\in N} x_{iv}L_{i,d(u)} \le t_{vu} \qquad \forall\, u \in U,\ \forall\, v \in K, \tag{6}$$

$$x_{iv} \in \{0,1\} \qquad \forall\, i \in N,\ \forall\, v \in K. \tag{7}$$

Objective function (1) minimizes the cost of placing each pair of virtual machines v and w at DCs i and j. Constraints (2) require that the number of VMs in each DC must not exceed the DC capacity. Constraints (3) require that each VM must be assigned to exactly one DC. Constraints (4) require that the given bandwidth between each pair i and j of DCs should not be surpassed by the total sum of bandwidth required among the virtual machines placed in these DCs. Constraints (5) assure that the latency required between each pair of VMs should be respected, i.e, if VMs v and w are placed respectively at DCs i and j, then the latency between DCs i and j should not exceed the maximum required latency between VMs v and w. Constraints (6) require that the latency between a VM v and the DC where the user u is located be respected, i.e, a VM v can be only placed at DC i if the latency between i and $d(u)$ is less than or equal to a given latency between the VM v and the user u. Finally, constraints (7) define the domain of the decision variables.

The performance of mixed integer linear solvers has improved considerably over the last few years. CPLEX[2] is a general-purpose black-box solver based on simplex and a branch-and-bound algorithm with the state-of-the-art exact algorithms for integer programming and has been successfully applied in many combinatorial optimization problems. To analyze the performance of CPLEX and provide baseline results for comparison of heuristic methods, the following subsections present two linear mathematical models for VMPlacement problem.

2.2. Linear mathematical model I: LMVMP

Based on the model from Lee and Ma (2004) for the GQAP, and from Frieze and Yadegar (1983) for the QAP, and introduced in Stefanello et al. (2015b), we present a mixed-integer

[2] `www.cplex.com`

linear model for the VMPlacement. Let $y_{ivjw} = x_{iv}x_{jw}$, $\forall\ i,j = \{1,\dots,N\}$ and $v,w = \{1,\dots,K\}$, the mathematical model referred to as LMVMP can be formulated as:

$$\min \sum_{i\in N}\sum_{v\in K} c_{iv}x_{iv} + \sum_{i\in N}\sum_{j\in N}\sum_{v\in K}\sum_{w\in K} y_{ivjw}C_{ij}b_{vw} \tag{8}$$

subject to:

$$\sum_{v\in K} x_{iv} \le a_i \qquad \forall\ i\in N, \tag{9}$$

$$\sum_{i\in N} x_{iv} = 1 \qquad \forall\ v\in K, \tag{10}$$

$$\sum_{i\in N} y_{ivjw} = x_{jw} \qquad \forall\ v,w\in K,\ \forall\ j\in N, \tag{11}$$

$$y_{ivjw} = y_{jwiv} \qquad \forall\ v,w\in K,\ \forall\ i,j\in N, \tag{12}$$

$$\sum_{v\in K}\sum_{w\in K} y_{ivjw}b_{vw} \le B_{ij} \qquad \forall\ i,j\in N, \tag{13}$$

$$\sum_{i\in N}\sum_{j\in N} y_{ivjw}L_{ij} \le l_{vw} \qquad \forall\ v,w\in K, \tag{14}$$

$$\sum_{i\in N} x_{iv}L_{i,d(u)} \le t_{vu} \qquad \forall\ u\in U,\ \forall\ v\in K, \tag{15}$$

$$x_{iv} \in \{0,1\} \qquad \forall\ i\in N,\ \forall\ v\in K, \tag{16}$$

$$0 \le y_{ivjw} \le 1 \qquad \forall\ i,j\in N,\ \forall\ v,w\in K. \tag{17}$$

Model LMVMP is obtained by replacing the product $x_{iv}x_{jw}$ by y_{ivjw} from QMVMP. In addition four extra sets of constraints are considered. Constraints (11) and (12) define the relation between variables x and y. Constraints (12) impose the symmetry relation to variables y. Finally, constraints (17) define the domain of variables y.

The model QMVMP has quadratic constraints, while LMVMP. The objective function is also linear. However, the mixed-integer linear problem LMVMP has a considerably larger number of variables, having variables y_{ivjw} in addition to the previous variables x_{iv}. Thus, the number of variables changes from $O(NK)$ in QMVMP to $O(N^2K^2)$ in LMVMP. We note that if the optimal solution of LMVMP is (x^*_{iv}, y^*_{ivjw}), then (x^*_{iv}) is the optimal solution of QMVMP. The proof that both models are equivalent can be obtained by extending the proof for QAP provided in Lee and Ma (2004).

2.3. Linear mathematical model II LMVMP-II

Next we present a second linearization for VMPlacement problem (LMVMPII), introduced in Stefanello et al. (2015b). This linear model is derived from Kaufman and Broeckx (1978) for QAP, which the linearization for QAP with a lower number of variables and constraints. In Lee and Ma (2004) the authors extend the formulation for GQAP. In this model, each binary decision variable x_{iv} is set to one when VM v is located at DC i, and zero otherwise. The auxiliary variables y_{iv} aggregate the cost for each placed VM v in a DC i, and n_{ijb} aggregate the bandwidth from VM v between data centers i and j.

$$\min \sum_{i\in N}\sum_{v\in K} c_{iv}x_{iv} + \sum_{i\in N}\sum_{v\in K} y_{iv} \tag{18}$$

subject to:

$$\sum_{v\in K} x_{iv} \le a_i \qquad \forall\, i \in N, \tag{19}$$

$$\sum_{i\in N} x_{iv} = 1 \qquad \forall\, v \in K, \tag{20}$$

$$\sum_{j\in N}\sum_{w\in K} C_{ij}b_{vw}x_{jw} - y_{iv} \le m_{iv}(1-x_{iv}) \qquad \forall\, v \in K,\ \forall\, i \in N, \tag{21}$$

$$\sum_{w\in K} b_{vw}x_{jw} - n_{ijv} \le M(1-x_{iv}) \qquad \forall\, v \in K,\ \forall\, i,j \in N, \tag{22}$$

$$\sum_{v\in K} n_{ijv} \le B_{ij} \qquad \forall\, i,j \in N, \tag{23}$$

$$L_{ij}x_{iv} \le l_{vw} + L_{ij}(2 - x_{iv} - x_{jw}) \qquad \forall\, v,w \in K, \forall\, i,j \in N, \tag{24}$$

$$\sum_{i\in N} x_{iv}L_{i,d(u)} \le t_{vu} \qquad \forall\, u \in U,\ \forall\, v \in K, \tag{25}$$

$$x_{iv} \in \{0,1\} \qquad \forall\, i \in N,\ \forall\, v \in K, \tag{26}$$

$$y_{iv} \ge 0 \qquad \forall\, i \in N,\ \forall\, v \in K, \tag{27}$$

$$n_{ijv} \ge 0 \qquad \forall\, v,w \in K,\ \forall\, i,j \in N. \tag{28}$$

where

$$m_{iv} \ge \sum_{j\in N}\sum_{w\in K} C_{ij}b_{vw}, \quad \forall\, i \in N,\ \forall\, v \in K.$$

Constraints (21) impose the cost between the data centers i and j to the variables y_{ij}. Constraints (22) and (23) impose the bandwidth constraints while constraints (24) impose the latency constraints. The constraints (24) can be replaced by

$$x_{iv} + x_{jw} \le 1 \quad \forall\, i,j \in N,\ \forall\, v,w \in K \text{ if } L_{ij} > l_{vw}. \tag{29}$$

We observe that CPLEX converts (24) into (29) in the pre-processing phase. Finally, constraints (26), (27), and (28) define the domains of the variables.

This model is not used in practice since it uses big-M constraints (21), and the root-node bound is always zero. However, a stronger formulation can be obtained by adding the following cuts

$$y_{iv} \ge Y_{iv}x_{iv} \quad \forall\, i \in N,\ \forall\, v \in K, \tag{30}$$

where Y_{iv} is defined as the optimal value of the following generalized assignment model:

$$Y_{iv} = \min \sum_{j \in N} \sum_{w \in K} C_{ij} b_{vw} x_{jw} \tag{31}$$

subject to:

$$\sum_{w \in K} x_{jw} \leq a_j \qquad \forall\, j \in N, \tag{32}$$

$$\sum_{j \in N} x_{jw} = 1 \qquad \forall\, w \in K, \tag{33}$$

$$\sum_{j \in N} \sum_{w \in K} b_{vw} x_{jw} \leq B_{ij} \qquad \forall\, w \in K, \tag{34}$$

$$(24),\ (25) \tag{35}$$

$$x_{iv} = 1 \tag{36}$$

$$x_{jw} \in \{0, 1\} \qquad \forall\, j \in N,\ \forall\, w \in K. \tag{37}$$

These additional cuts were applied for the 3-dimensional assignment problem in Mittelmann and Salvagnin (2015). Note that these cuts can also be applied for the GQAP suppressing the constraints (22)–(25) and (28) from LMVMPII, and (34)–(35) from the assignment model, which are the specific constraints for VMPlacement.

3. Heuristic Approaches

In this section we propose two heuristics approaches to solve the VMPlacement problem. We first describe the local search procedures and two path-relinking strategies used in the proposed metaheuristics. Finally, we describe the Greedy Randomized Adaptive Search Procedure (GRASP), followed by the Biased Random-Key Genetic Algorithm (BRKGA).

Placing a virtual machine in a data center can violate some constraints. To deal with this situation, we use a penalization strategy to minimize the number of violated constraints. Thus, the cost of placing a VM v in DC i is calculated by the regular placement cost in addition to a sufficiently large number M for each violated constraint. This penalization strategy is applied whenever a solution is evaluated. In our experiments we use $M = 10^{10}$. Also, the notation for a solution S represents a vector of size $|K|$ with $S = \{n_1, \ldots, n_{|K|}\}$, with n_v representing the label (or index) of the data center where the virtual machine v is placed.

3.1. Local search procedures

Local search is a general approach for finding and improving solutions to hard combinatorial optimization problems. The most basic strategy of local search algorithms is to start from an initial solution and iteratively replace the current solution by a better neighbor solution, until no improvement is possible. A neighbor solution can be obtained by applying moves defined by a neighborhood structure. In this paper we define four neighborhood structures to obtain neighbor solutions, namely *shift*, *swap*, *chain2L*, and *chain3L*.

Shift. A shift operation moves a virtual machine from the current data center to a different data center. In a *shift search*, we select a virtual machine v in a circular order of their indexes (starting from index zero), and calculate the cost to move it from the current data center to each other data center. A virtual machine v is moved to the data center that produces the greatest improvement in the objective function. This procedure is repeated until reaching a local minimum.

Swap. A swap operation interchanges the positions of two virtual machines. In a *swap search*, we evaluate the cost of all swap moves between two virtual machines v and w in a circular order of their indexes (starting from index zero). When an improvement is reached, the virtual machine positions are interchanged, and the search continues by selecting a next pair of virtual machines. The procedure ends when no swap move can improve the solution. The evaluation of symmetrical movements is avoided.

Chain2L. A chain operation is a composition of two shift moves applied sequentially. In a *chain2L search*, the first shift move selects a virtual machine v from the data center i to move to a data center j. In the second shift move, a virtual machine w from j is moved to a data center k. We evaluate all compositions of two shift moves. When an improvement is reached, the moves are applied to the solution and the process is restarted with the new solution. Note that this neighborhood includes the *swap search* when k is equal to i.

Chain3L. The last neighborhood proposed extends the concept of the chain moves to a composition of three shift moves applied sequentially. A *chain3L search* involves four virtual machines and up to four data centers. This search also comprises three shift moves between two data centers, or a circle move among three data centers.

Given a solution S, the worst case time complexity to evaluate the cost to insert or remove a virtual machine v in S is $O(|K|)$. This complexity is reached because it is necessary to evaluate whether the latency requirement is satisfied between v and $w \in S$. Also, it is necessary to evaluate the bandwidth cost between v and $w \in S$. Capacity constraints, uniqueness in the placement, and user latency requirements can be evaluated in constant time. Therefore, for the *shift search* $|K|$ removals and $|K| * |N|$ insertions are evaluated, resulting in a complexity of $O(|K|^2|N|)$ (for the case of no improvement during the search). For the *swap search* the time complexity is $O(|K|^3)$, while for the *chain2L search* it is $O(|K|^3|N|)$. Finally, the *chain3L search* has complexity of $O(|K|^4|N|)$.

Naturally, in all the searches described above, the processing time can be reduced avoiding the evaluation of moves that lead to a worse solution. Also, heuristic strategies do not need to explore the whole neighborhood, and only evaluate a subset of the neighbor solutions. This is done in the *chain3L search* by prohibiting the insertion of a virtual machine in some data centers, thus reducing the search space.

Two heuristic strategies to reduce the amount of explored neighbor solutions in the *chain3L search* are proposed. The first reduction heuristic strategy is based on the cost Y_{iv} described in the Model (31)-(37) from Subsection 2.3. Given a parameter $\alpha_{3L} \in [0, 1]$, we allow to insert a virtual machine v in a data center i only if i belongs to the $\lfloor N * \alpha_{3L} \rfloor$ data centers with the lowest cost Y_{iv}. A low value of α_{3L} indicates a restricted search only in data centers with low values of Y_{iv}. A high value of α_{3L} indicates a more broad search. An analysis for the instances described in Section 4.1 with their respective best known solutions corroborates this for this heuristic strategy. In these solutions, more than 50% of the virtual machines are placed in the 30% data centers with lowest cost Y_{iv}, and approximately 75% are placed in the 50% data centers with the lowest cost.

The second reduction heuristic strategy is based on the communication cost assigned to the data centers. Given a solution $\mathcal{S}$, let C_i be the sum of the communication cost from i to all other data centers. We only evaluate compositions of movements that start from a virtual machine that belongs to the β_{3L} data center with the highest cost C_i. In summary, the idea is to limit the search to movements that reduce the communication cost of the most required data centers. Combining both reduction heuristics, the search space is considerably smaller, and the heuristics are still able to visit solutions in a promising search space. By default, we use $\alpha_{3L} = 0.3$ and $\beta_{3L} = 3$.

The neighborhoods are integrated based on the idea of Variable Neighborhood Descent (Hansen et al., 2010). Every neighborhood is applied sequentially starting from the lowest to highest complexity neighborhood (*shift*, *swap*, *chain2L*, and *chain3L*). However, the search is restarted with the first neighborhood if an improvement is reached. We name each local search strategy using two characters. The first character is a number that indicates the highest complex neighborhood applied. The character `V` indicates that we use the VND strategy to integrate the neighborhoods. For example, the name `3V` indicates that we consider local search procedures *shift search*, *swap search*, and *chain2L search*.

Note that using a penalization strategy described in the previous subsection, the local search is also applied to infeasible solutions. In this case, the local search also works as a repair procedure.

3.2. Path-relinking

Path-relinking (PR) is an approach to integrate intensification and diversification in the search. It consists in exploring trajectories that connect high-quality solutions. Starting from an initial solution, the scheme moves from one solution to another until the target solution is reached. The objective consists in finding a solution that is better than both the initial and target solutions.

Path-relinking was first suggested for tabu search in Glover (1989) and then formalized in Glover and Laguna (1993). Since then, this strategy was applied on a large number of combinatorial optimization problems and related studies as Glover (1997), Glover et al. (2000), Resende and Ribeiro (2005), Resende et al. (2010), Festa and Resende (2013), and Glover (2014), just to name a few.

Algorithm 1 shows a pseudo-code for the path-relinking operator between solutions S and T, where S is the initial solution and T is the guiding solution. In line 2 the incumbent solution is initialized. The loop between lines 3 and 10 is repeated while the distance between both solutions is greater than an input parameter δ_{lim}. The distance $\Delta\{S,T\}$ is the minimum number of moves needed to transform S into T or vice-versa. We use $\delta_{lim} = \Delta\{S,T\}/2$ defined at the beginning of the procedure, since we observed that the probability of finding an improved solution is greater in the first steps of the path. In line 4 a movement is applied to the solution S in the direction of solution T. Basically, we analyze all changes in S to T that reduce the distance $\Delta\{S, T\}$ by one, and apply in S the change with the least cost.

Several studies have experimentally found that it is convenient to add a local search exploration from some of the generated solutions in the path (Martí et al., 2006). Since two consecutive solutions obtained by a relinking step are often very similar, it is generally not efficient to apply the local search at every step of the relinking process. Thus, in line 7 the local search is applied at each n iterations, where $n = (|K| * 0.5)/(4 + |K|/50)$. The parameter n is rounded up to the nearest even number. This ensures that the local search is applied to S and T each time, since, in line 10, the solutions are interchanged to use a *back-and-forward* strategy (Festa and Resende, 2013). Finally, in line 8 the incumbent solution is updated and returned in line 11.

Algorithm 1: Pseudo-code for a greedy path-relinking operator.

```
1  Procedure PathRelinkingOperator($S$,$T$)
2      $S^* \leftarrow S$;
3      while $\Delta\{S, T\} \geq \delta_{lim}$ do
4          $S \leftarrow$ MakeMovement$(S, T)$;
5          $S' \leftarrow S$;
6          if $LSCondition$ then
7              $S' \leftarrow$LocalSearch($S'$);
8          if $f(S') < f(S^*)$ then
9              $S^* \leftarrow S'$;
10         Swap$(S, T)$;
11     return $S^*$;
```

Algorithms 2 and 3 describe two frameworks of how to manage the elite set $\mathcal{E}$, and how solutions are selected for the path-relinking operator. In the first approach, the path-relinking operator is applied between a provided solution S and a selected solution T from the elite set. In the second approach, the path-relinking operator is applied multiple times, between a provided solution S and every solution from the elite set.

To simplify the notation, we refer to path-relinking the general process to manage the elite set and apply the path-relinking operator between the two solutions. With respect to the specific elite-set management procedures, we call the first approach PRS and the second approach PRM. The first two letters indicate that the algorithm is related to the path-relinking and the last letter indicates the framework for managing the path-relinking operator. In the first, S indicates the application of a single path-relinking operator and in the second, M indicates the application of a multiple path-relinking operator.

In the beginning of Algorithm 2, the elite set $\mathcal{E}$ is empty. The maximum size δ_{max} of the elite set is an input parameter. A solution S is added to $\mathcal{E}$ if it improves the best solution or is better than the worst elite solution and is sufficiently different from each solution in $\mathcal{E}$ (lines 3, 6, and 11). We define that two solutions are sufficiently different if $\Delta\{S, T\} > \delta_{dis}$. We denote by $S \not\approx \mathcal{E}$ if $\Delta\{S, T'\} > \delta_{dis}, \forall\ T' \in \mathcal{E}$. In our case, we use $\delta_{dis} = \lfloor |K| * 0.1 \rfloor$. If the elite set has a minimum number of solutions ($\delta_{min} = 2$ in line 2), and S is sufficiently different, a solution T is selected for the path-relinking operator between S and T.

In line 4, solutions are ordered by a linear rank $r(T_i)$, $\forall\ i = 1 \dots |\mathcal{E}|$, where the best solution has rank $r(T_1) = |\mathcal{E}|$ and the worst solution has rank $r(T_{|\mathcal{E}|}) = 1$. A solution $T \in \mathcal{E}$ is selected using the roulette wheel criterion, i.e, T is selected from the elite set with probability $r(T)/\sum_{i \in r(T_i)}$. With this criterion, better solutions have more chance to be chosen. After T is selected, the path-relinking operator is applied between S and T.

Lines 6 to 9 update the elite set after the path-relinking operator is applied. Line 6 evaluates whether S is sufficiently different or improves the best solution. If this is the case, S is added to $\mathcal{E}$. Line 8 maintains the cardinality of the elite set. T' is the most similar solution to S with worst solution than S, i.e, $T' \leftarrow \{T'' \in \mathcal{E} \mid f(T'') > f(S)$ and $\Delta\{T'', S\} \leq \Delta\{T'', S'\}, \forall\ S' \in \mathcal{E}\}$. Finally, in line 13, solution S is returned.

Algorithm 2: Pseudo-code for path-relinking management framework `PRS`.

```
1  Procedure PathRelinking($S$)
2      if $|\mathcal{E}| > \delta_{min}$ then
3          if $S \not\approx \mathcal{E}$ then
4              Select $T \in \mathcal{E}$ ;
5              $S \leftarrow$PathRelinkingOperator($S, T$);
6              if $S \not\approx \mathcal{E}$ then
7                  $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
8              if $|\mathcal{E}| > \delta_{max}$ then
9                  Remove $T' \in \mathcal{E}$;
10     else
11         if $S \not\approx \mathcal{E}$ then
12             $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
13     return $S$;
```

Another approach is outlined in Algorithm 3. The main difference from the previous approach is that when a solution S is added to $\mathcal{E}$, the path-relinking operator is applied between all pair of solutions from $\mathcal{E}$ to which the operator was not previously applied. This is done in the loop in lines 5 to 13. The flag $hasPairsST$ can be easily updated using memory structures, and it indicates whether there are S' and T' in the elite set for which the path-relinking operator was not applied. In line 6 two solutions are selected. Pairs with better objective functions are selected first. The insertion operation (line 11) and the removal operation (line 13) follow the same rules as in the previous algorithm. In line 14 a solution is removed from the elite set in case it is full. We also use a roulette wheel criterion by rank, where the best solution has rank equal to zero, and the worst solution has rank $|\mathcal{E}| - 1$. This ensures that the best solution are kept in the pool, and the worst solution has the highest probability of being removed. Finally, in line 18 solution S is returned.

Algorithm 3: Pseudo-code for path-relinking management framework `PRM`.

```
1  Procedure PathRelinking($S$)
2      if $|\mathcal{E}| > \delta_{min}$ then
3          if $S \not\approx \mathcal{E}$ then
4              $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
5              while $hasPairsST$ do
6                  Select $S'$ and $T'$;
7                  $S' \leftarrow$PathRelinkingOperator($S'$,$T'$);
8                  if $f(S') < f(S)$ then
9                      $S \leftarrow S'$;
10                 if $S' \not\approx \mathcal{E}$ then
11                     $\mathcal{E} \leftarrow \mathcal{E} \cup \{S'\}$;
12                 if $|\mathcal{E}| > \delta_{max}$ then
13                     Remove $T' \in \mathcal{E}$;
14             Remove $T' \in \mathcal{E}$;
15     else
16         if $S \not\approx \mathcal{E}$ then
17             $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
18     return $S$;
```

Note that Algorithms 2 and 3 are designed with independent components, requiring only few configuration parameters (as local search strategy and elite size), and a procedure that provides different solutions. For this reason, these path-relinking strategies can be easily included in different heuristic frameworks. In the next subsections, we describe the GRASP and BRKGA metaheuristics which use the path-relinking strategies.

3.3. Greedy Randomized Adaptive Search Procedure - GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) is a multi-start metaheuristic for combinatorial optimization problems (Feo and Resende, 1989; 1995; Resende and Ribeiro, 2010; Martí et al., 2013; Resende and Ribeiro, 2014; 2016). The algorithm is basically composed of two phases: construction and local search. The construction phase builds a feasible solution following a greedy randomized criterion scheme, whose neighborhood is investigated until a local minimum is found during the local search phase. These phases are repeated over the iterations and the best local optimum found is returned as the heuristic solution. Its basic implementation is memoryless, because it does not make use of information collected in previous iterations. One way to add memory to GRASP is its hybridization with path-relinking. A large review of this technique is presented in Festa and Resende (2013).

A high-level description of GRASP is presented in Algorithm 4. GRASP iterations are carried out in lines 3 to 8. In line 4, the procedure attempts to build a greedy randomized solution (see Algorithm 5). A solution S is used as the starting solution for the local search in line 5. In line 6 path-relinking is applied if a hybrid approach is used. If the local optimum S is better than the incumbent solution, then, in line 8 the incumbent solution is updated. In line 9, the best solution found overall GRASP iterations is returned as the GRASP solution.

Algorithm 4: Pseudo-code for GRASP.

```
1  Procedure GRASP()
2      $S^* \leftarrow \emptyset$;
3      while stopping criterion is not satisfied do
4          $S \leftarrow$ GreedyRandomized();
5          $S \leftarrow$ LocalSearch($S$);
6          $S \leftarrow$ PathRelinking($S$);
7          if $f(S) < f(S^*)$ then
8              $S^* \leftarrow S$;
9      return $S^*$;
```

Algorithm 5 shows a general pseudo-code for the greedy randomized constructive heuristic used in the GRASP. The construction builds a solution S, one element at a time. In line 2, solution S is initialized empty. In line 3, the set of candidate elements is initialized. In line 4, the cost of placing each candidate element in a data center is calculated. The solution is built in the loop in lines 5 to 10. In line 6, a restricted candidate list (RCL) is built, and in line 7 a candidate is selected. In line 8 this element is added to the partial solution. In line 9, the candidate is removed from the set of candidates $\mathcal{C}$. In line 10 the placement costs for each candidate are re-evaluated. Finally, in line 11, solution S is returned.

Algorithm 5: Pseudo-code for a greedy randomized constructive solution procedure.

```
1  Procedure GreedyRandomized()
2      $S \leftarrow \emptyset$;
3      Initialize set of candidates $\mathcal{C}$;
4      Evaluate the incremental cost of candidates;
5      while $\mathcal{C} \neq \emptyset$ do
6          Build the $RCL$;
7          Select $c \in RCL$;
8          Add $c$ to solution: $S \leftarrow S \cup \{c\}$;
9          Update candidates: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$;
10         Re-evaluate the incremental costs;
11     return $S$;
```

Based on this framework, four constructive heuristics were developed to generate initial solutions in line 4 of GRASP.

Random. In the first constructive heuristic, at each placement step a yet unplaced virtual machine is randomly selected and placed in a random data center. Note that this constructive approach is not a greedy heuristic. We decided to evaluate this approach since it is one of the simplest constructive methods, and we use it as a baseline for comparing more sophisticated constructive methods.

Greedy. In the second constructive heuristic, at each placement step a yet unplaced virtual machine is randomly selected and placed in the data center that produces the least increase in the objective function. In this case, the RCL is composed of only one candidate that is the best data center for a specific randomly selected virtual machine.

DC-Greedy. In the third constructive heuristic, on each placement step a yet unplaced virtual machine is randomly selected and placed in one of the n data centers from the RCL. The RCL contains the candidates that produce the lowest incremental placement cost. By default, we use $n = \max\{3, |N| * 0.2\}$. The value of n can vary at each iteration since we consider some additional conditions. First, candidates with the same cost as the first n candidates are also added to the RCL. Second, if at least one candidate can be added to the solution keeping it feasible, then candidates that generate infeasibilities are ignored.

VM-Greedy. In the last constructive heuristic, at each placement step, all possible placements of virtual machines to data centers are evaluated, and one of the n best candidates is randomly selected. In this constructive heuristic, we use $n = \max\{5, |K|{*}0.1\}$ for the RCL size, but the effective value of n can vary since we also include both additional conditions described in the previous constructive heuristic.

In these constructive heuristic, any random selection is based on a uniform distribution. Note that these constructive heuristic can generate infeasible solutions, and the repair process is done by the local search procedure.

3.4. Biased random-key genetic algorithm - BRKGA

A biased random-key genetic algorithm (BRKGA) is a populational metaheuristic that recently has been successfully applied to several hard combinatorial optimization problems (Gonçalves and Resende, 2011). A BRKGA is a class of genetic algorithms in which solutions are encoded as vectors of random keys, i.e. randomly generated real numbers from uniform distribution in the interval $[0, 1)$. A vector of random keys is translated into a solution of the optimization problem by a *decoder*. A decoder is a deterministic algorithm that takes as input a vector of random keys and returns a solution of the optimization problem as well as its cost (or *fitness*).

Algorithm 6 presents a general scheme of a BRKGA. The algorithm starts with a set of p random vectors of size n (*population*). Parameter n depends on the encoding while parameter p is user-defined. Starting from the initial population, the algorithm generates a series of populations. Each iteration of the algorithm is called a *generation*. The algorithm evolves the population over the generations by combining pairs of solutions from one generation to produce offspring solutions to the following generation.

At each generation g, the decoder is applied to all newly created random keys, and the population is partitioned into a smaller set of p_e elite solutions, i.e., the best fit p_e solutions in the population, and another larger set of $p - p_e > p_e$ non-elite solutions (line 6). Population $g{+}1$ is generated as follows. All p_e elite solutions of population g are copied without changing to population $g + 1$ (line 7). This elitist strategy maintains the best solutions on hand. To

Algorithm 6: Pseudo-code of BRKGA.

```
1  Procedure BRKGA()
2      Generate $p$ vectors of random keys;
3      Decode each vector of random keys;
4      while stopping criterion is not satisfied do
5          Sort solutions by their fitness;
6          Classify solutions as elite and non-elite;
7          Copy elite solutions to next population;
8          Generate mutants in the next population;
9          Combine an elite and a non-elite and add offspring to next population;
10         Decode each vector of random keys;
11     return best solution $S$;
```

ensure that mutation is present in the evolution, p_m mutant are added to population $g+1$. A *mutant* is simply a vector of random keys, generated in the same fashion as the initial solutions (line 8).

With $p_e + p_m$ solutions accounted for population $g+1$, other $p - p_e - p_m$ additional solutions must be generated to complete the p solutions that make up population $g+1$. This is done through *mating* or *crossover* (line 9). A parent-A is selected randomly from the elite solutions, and a parent-B is selected randomly among the set of non-elite solutions. A child C is produced by combining the parents using parameterized uniform crossover (Spears and DeJong, 1991). Let $\rho_A > 1/2$ be the probability that the offspring solution inherits the key of parent-A and $\rho_B = 1 - \rho_A$ be the probability that it inherits the key of parent-B, i.e. $c_i = a_i$ with probability ρ_A or $c_i = b_i$ with probability $\rho_B = 1 - \rho_A$, where a_i and b_i are, respectively, the i-th key of parent-A and parent-B, for $i = 1, \ldots, n$.

Random key genetic algorithms were first introduced in Bean (1994). BRKGA differs from the original approach in the way parents are chosen from the population and how the child inherits the key (Gonçalves and Resende, 2011). In that work, the authors describe a BRKGA as having *problem-independent* and *problem-dependent* components. The independent components are applied without any knowledge of the problem. The problem-dependent components are the only connection with the problem. Thus, to describe a BRKGA, one needs only to show how solutions are encoded and decoded, what choice of parameters p, p_e, p_m, and ρ_A was made, and how the algorithm stops. We next describe the encoding and decoding procedures for the proposed algorithm, and attribute values for parameters and stopping criterion in Section 4.

3.4.1. Decoders

Solutions of the optimization problem are encoded as a vector $\mathcal{X}$ with $n = |K|$ random keys. To translate this vector into the solution of the VMPlacement problem, we propose two decoders, as described next.

Greedy Ordered Decoder - D1: In this decoder, the keys provide the order of placement. Following this order, a greedy strategy is used, placing each VM at the DC which produces the least increase in the objective function.

The decoder starts with a list in which each element is composed of the random key and the index of the virtual machine. The list is sorted by the keys. Now, the sorted list of indices of virtual machines is used as an order in which virtual machines are placed. Following this order, the next step is to place each virtual machine v at a DC. This is done by placing virtual machine v at the DC i which produces the least increase in the objective function. Note that the cost to insert VM v in each DC considers the previous virtual machines placed. When all VMs are placed, the decoder returns the fitness value for the vector $\mathcal{X}$ of random keys.

Location Decoder - D2: In this decoder, each key is decoded as the data center in which the virtual machine should be placed. Let k_i be the key corresponding to the VM of index i in $\mathcal{X}$, then this decoder simply places the VM of index i at DC $\lfloor k_i * |N| \rfloor$. Figure 2 shows an example of decoding for decoder D2 for three data centers, eight virtual machines and a vector $\mathcal{X}$ of random keys.

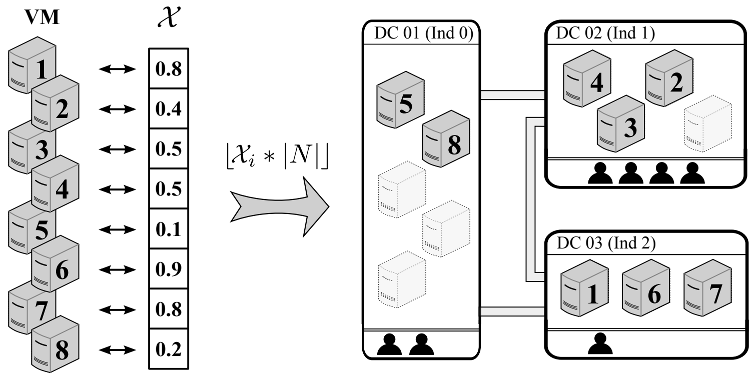


Figure 2: Example of decoding for decoder D2.

Since both decoders are deterministic, we always obtain the same solution S' from a vector of random keys $\mathcal{X}$. In the case where a deterministic local search strategy is applied to S', a solution S'' can be obtained using the decoders and the local search, and thus, S'' can be associated with $\mathcal{X}$. However, depending on the decoder, its is possible to use a process called *recode* in $\mathcal{X}$, to obtain S'' only by using the decoder (without local search). In decoder D1, the recode can be computationally expensive so we decided to maintain the local search as part of the decoder. In decoder D2, the recode can be applied by calculating the value of the key that will correspond to a data center in S''. Let i' the index of DC i, $lb = i'/|N|$ and $lu = (i+1)'/|N|$, a key that corresponds to DC can be given by $lb + (lu - lb)/2$ or any number in the interval $[lb, ub)$. Experiments including the recode process are presented in the Subsection 4.4. To simplify the notation, we denote by D3 the decoder D2 with recode process.

3.4.2. Hybrid BRKGA and path-relinking

Path-relinking is an approach to integrate intensification and diversification in the search and can be incorporated on many metaheuristic frameworks. Path-relinking is frequently used with GRASP algorithms (Laguna and Martí, 1999; Oliveira et al., 2004; Mateus et al., 2010; Festa and Resende, 2013), but references can be found for other metaheuristics as Scatter Search (Glover et al., 2003), Tabu Search (Armentano et al., 2011), VNS (Festa et al., 2002). Regarding GAs, we find several papers where the path-relinking technique is applied, as for example Basseur et al. (2005) and Vallada and Ruiz (2010), just to name a few.

In this paper, we propose a new approach that hybridizes path-relinking with a BRKGA algorithm. In our approach, we include the Algorithms 2 or 3 every n generations of BRKGA. By default we use $n = 1$, i.e., we apply the path-relinking each time a new generation is produced. Since both path-relinking procedures require a solution S, we randomly select S with a uniform distribution among all elite solutions of the BRKGA population. In the case that the path-relinking returns a solution S' that is better than S, then the corresponding vector $\mathcal{X}$ of S is recoded to be adjusted to S'.

In Section 4, experiments with both path-relinking strategies in comparison with the standard approach are presented in detail. Since the path-relinking uses the recode process, experiments are performed only with decoder D3.

4. Computational results

The experiments were conducted on a computer with an AMD FX-8150 Eight-Core 3.6 GHz CPUs, with at least 32 GB of RAM running GNU/Linux, except for experiments with CPLEX which we used a computer with quad-core Intel Xeon E5530 2.4 GHz CPUs, with at least 48 GB of RAM running GNU/Linux. Algorithms are implemented in `C++`, with optimization flag `-O3`. For BRKGA we use the API described in Toso and Resende (2015). The commercial solver IBM ILOG CPLEX Optimizer version 12.6.0.0 (C++ API) was used to evaluate the mathematical models. All experiments used a single thread but multiple experiments were run in parallel.

Experiments were conducted with two main objectives. The first was to investigate the performance of CPLEX considering the different mathematical models described in Section 2. The second was to evaluate the performance of the heuristic approaches described in Section 3 and some hybridized variants, including heuristic approaches, local search strategies, and path-relinking procedures.

Initially we describe the method to generate the dataset used in the experiments. Following that, we report results for CPLEX, GRASP, BRKGA, and then we report the best results of each heuristic method. Finally, we use our best approaches on a set of instances of GQAP, comparing our results with the state-of-the-art results for the problem described in Mateus et al. (2010).

4.1. Data set

In this subsection we present a problem-instance generator used to create the data set used in the computational experiments reported by this paper. Since we do not have access to real data, we randomly generate a set of instances with some empirically defined parameters to study as many different scenarios as possible.

For each instance the generator receives as input four parameters: $|N|$, $|K|$, $|U|$, and P (the last parameter represents the percentage of the overall data center occupation).

To ensure the generator creates instances that admit feasible solutions, we generate the data for each instance based on n sets of pre-placed virtual machines to data centers (by default $n = 3$). Given the capacity of each data center, each set of pre-placed $s \in \mathcal{S}$ is generated by randomly placing each virtual machine at a data center with probability proportional to the data center capacity, ensuring the capacity is not violated. Biased on these pre-placements, we generate the remaining data respecting the constraints of the problem, ensuring at least n feasible solutions for each instance.

Considering a random number generator using uniform distribution, and M' a sufficiently large number, we generate the parameter data for each instance using the following steps.

Data center capacity: The total number of available virtual machines is given by $n' = \max\left(|N|, \left\lceil \frac{|K|}{|P|} \right\rceil\right)$. Thus, to define the values of a_i for each DC i, we start with all $a_i = 0$ and select n' times a random data center i, and increase a_i by one. We also ensure that each data center has a capacity a_i greater or equal to one. At this step, the n pre-placements described before are generated.

Required virtual machine bandwidth: For each pair of virtual machines $v \in K$ and $w \in K$ the bandwidth b_{vw} is a random number in the interval $[0 : 9]$. This matrix is symmetric, i.e, $b_{vw} = b_{wv}$, and $b_{vv} = 0$.

Data center bandwidth: Having defined the bandwidth between each pair of virtual machines in the previous step, we generate the values of data center bandwidth based in the pre-placements $\mathcal{S}$. Let b^s_{ij} be the sum of bandwidth between all virtual machines pre-placed to i and j in $s \in \mathcal{S}$. For each pair of data centers ij, we associate a bandwidth $B_{ij} = \max\{b^s_{ij}\}$, $\forall s \in S$. This matrix also is symmetric, i.e $B_{ij} = B_{ji}$, with $B_{ii} = M'$.

Data center latency: For each pair of data center ij, the latency L_{ij} is a random number selected in the interval $[5 : 20]$. This matrix is symmetric, i.e, $L_{ij} = L_{ji}$, with $L_{ii} = 0$.

Required virtual machine latency: Let l^s_{vm} be the latency L_{ij} between the data centers ij where v is placed in i and w is placed in j in the pre-placement $s \in \mathcal{S}$. We randomly select $n = |K| * 2$ distinct pairs vw to associate a required latency $l_{vw} = \max\{l^s_{vm}\}$, $\forall s \in \mathcal{S}$. The remaining latency l_{vw} is defined as M', indicating that no latency is required. We also ensure that $l_{vw} = l_{wv}$, and $l_{vv} = 0$.

Users in data centers: Users are allocated at random to data centers chosen with probability proportional to their capacity. More than one user can be located at the same data center.

Required user latency: For each user, we randomly select a virtual machine v to define a required latency. Let $i(s)$ be the data center where v is placed in $s \in \mathcal{S}$, thus the required latency between u and v is given by $t_{vu} = \max\{L_{d(u),i(s)}\}, \forall s \in \mathcal{S}$. The remaining user required latency t is set to M'.

Transferring data center cost: For each pair of data center ij, the cost C_{ij} is a random number in the interval $[10.00 : 100.00]$. This matrix is symmetric, i.e $C_{ij} = C_{ji}$, and $C_{ii} = 0$.

The parameters $|N|$ and $|K|$ can be used to define the instance size, while parameters $|U|$ and P can be used to adjust how the problem should be restricted. Finally, we generate a set of instances by combining values from $N = \{5, 10, 25\}$, $K = \{15, 20, 25, 50, 100, 150, 200\}$, $U = \{K_i * 0.5, K_i, K_i * 1.5\}$, and $P = \{70, 90\}$. Tables 1 and 3 list all instances we generated, and the values of $|N|$, $|K|$, $|U|$ and P are encoded in the name of instance in this respective order. All instances and their best known solutions are available at `www.inf.ufsm.br/~stefanello/instances/vmplacement`.

The cost c_{iv} is a generic user-defined parameter that can be used to consider a placement. This cost can be related to a simple requesting virtual machine or a setup time to start a virtual machine in a data center. We consider this parameter in the objective function for two reasons: i) To keep the problem as generic as possible, considering these additional costs. ii) Since we define the VMPlacement problem is a generalization of GQAP, we included this cost inherited from the generic definition of GQAP. However, in our experiments, without any loss of generality, we disregard this cost by simply set the cost to zero.

4.2. CPLEX results

In this section we investigate the performance of CPLEX for both linear mathematical models described in Section 2. We carried out an empirical study to analyze which size of instances the CPLEX solver can handle and prove optimality, and which model provides better integer solutions and lower bounds when CPLEX is terminated with a time limit. The objective is also to obtain baseline results for comparison of heuristic methods. We first analyze the results for small-size instances, followed by a set of the medium-size instances and large-size instances. We run CPLEX for the QMVMP model. Since the performance of the linear models was considered better than for the quadratic model, we report only results for the mixed-integer linear model.

4.2.1. Results for small size instances

In the first experiment, we evaluated the performance of CPLEX with the mathematical models described in Section 2. We used the standard CPLEX 12.6 solver for models LMVMP and LMVMPII. The time limit was set to three hours per run (10,800 seconds) and the number of threads to one. The remaining parameters were kept to their default values.

In Fischetti and Monaci (2014) the authors exploited erraticism in search and how to take advantage of this behavior. Also, the authors show that CPLEX can have a wide contrast in its behavior due randomized initial conditions. Thus, to better evaluate the CPLEX performance we perform ten runs for each instance, each one with a different random seed defined by the CPLEX parameter `RandomSeed`.

Table 1 shows CPLEX results. The first column shows the name of the instances. The second column (BKS) shows the objective function of the best known solution value for each instance (optimal solutions are showed in boldface). The next two columns show the average running times of CPLEX to solve each instance for each mathematical model. Numbers in parenthesis denote the number of runs that the solver did not prove optimality within the time limit. The next two columns show the average running times that CPLEX spent to find the BKS value. The average is over ten runs or the number of times that CPLEX found BKS (indicated by the number in parenthesis). A signal '−' indicates that no run found a solution as good as BKS within the time limit. Finally, the last two columns show the best running times over all runs that CPLEX spent to find the BKS.

Table 1: CPLEX detailed results for small instances.

		AVG Time(s)		**AVG BKS Time(s)**		**MIN BKS Time(s)**	
Instance	**BKS**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**
`05_015_007_70`	**25,844.02**	2.9	6.2	2.3	2.2	1.1	0.4
`05_015_007_90`	**23,557.30**	26.0	61.9	17.4	15.4	10.2	1.4
`05_015_015_70`	**10,904.78**	0.4	1.0	0.3	0.4	0.2	0.3
`05_015_015_90`	**24,354.96**	4.2	6.9	3.1	1.8	2.3	0.8
`05_015_022_70`	**14,163.60**	0.2	0.7	0.1	0.1	0.1	0.0
`05_015_022_90`	**32,318.02**	22.4	21.3	21.2	5.2	12.7	0.9
`05_020_010_70`	**38,572.62**	292.8	701.7	232.4	138.1	6.4	1.2
`05_020_010_90`	**64,710.80**	68.3	107.3	58.5	24.7	31.2	1.9
`05_020_020_70`	**55,288.76**	134.5	437.6	100.9	212.4	10.5	37.8
`05_020_020_90`	**57,574.90**	1.7	2.1	1.3	0.4	0.8	0.4
`05_020_030_70`	**28,433.34**	11.1	17.4	7.1	3.7	2.8	0.4
`05_020_030_90`	**66,088.70**	1.9	2.6	1.8	0.7	1.2	0.3
`05_025_012_70`	**43,300.76**	1271.5	4,025.5	1,063.8	862.6	195.0	2.2
`05_025_012_90`	100,865.02	10,800 (10)	10,800 (10)	9,902.6 (4)	6,165.6 (8)	9,746.0	28.7
`05_025_025_70`	**42,890.40**	58.4	126.7	52.5	39.0	26.9	1.3
`05_025_025_90`	103,791.96	10,800 (10)	10,800 (10)	9,764.3 (1)	–	9,764.3	–
`05_025_037_70`	**97,335.12**	161.4	592.8	117.0	161.2	19.6	2.8
`05_025_037_90`	**73,363.56**	10,262 (7)	10,800 (10)	8,993.2	3,782.8 (9)	5,916.3	33.8

We can draw three main observations from this experiment. First, for both models, the solver tends to increase significantly the time spent to prove optimality for instances with 5 or more data centers, and 25 or more virtual machines. This indicates a baseline for the size of instances that this version of CPLEX can handle and prove optimality using these models in this computer. The second observation is that CPLEX performance for both models was relatively similar. CPLEX was able to prove or not the optimality in the same instances, except for instance `05_025_037_90` which CPLEX proved optimality in three runs for LMVMP, while for LMVMPII CPLEX has an average gap of 8%. However, considering the time that each model spent to find a solution with objective function equal to BKS, in most cases CPLEX with LMVMPII spent less time than with LMVMP. Finally, the last observation is about the variance of time to find the BKS. With the randomized initial conditions, CPLEX presented a large variance in its behavior. For example, for the instance `05_025_012_70` and LMVMPII, in the best case CPLEX found a BKS in 2.2 seconds, while in the worst case it took $3,458.01$ seconds to find the same solution. For this instance, the time to prove optimality also has large variance, alternating between a minimum of $3,685.67$ and a maximum of $4,828.05$ seconds.

We also evaluate the model LMVMPII without including the additional set of cuts (30). In comparison with the model with the cuts, we observed a worse performance of CPLEX.

For six instances CPLEX was not able to prove optimality in any of the ten runs. For these cases, the gap of CPLEX is high. For example, for the instance `05_025_012_90`, the average gap was 53%, confirming that the relaxation quality of this model is low. Furthermore, the time to find a BKS increased by a factor of about 8, and the number of nodes explored within the time limit or to prove optimality increased by a factor about 7 times in comparison with the model that includes the set of cuts (30).

4.2.2. Results for medium and large size instances

In this subsection we present results for CPLEX when applied to a set of medium and large size instances. In this experiment, we evaluate CPLEX for each model and each instance with one run for a time limit of one day (86,400 seconds).

Table 2 shows for each instance the BKS obtained over all experiments, including the value reported in this paper for heuristic methods and additional non-reported experiments used to evaluate the previous version of the algorithms. The column FO shows, for each model, the objective function value of the best integer solution found by CPLEX. Column CPLEX GAP shows the gap returned by CPLEX. The last column shows the gap from BKS to the best integer solution returned by CPLEX.

The first observation from this experiment is that for instances with 10 data centers, CPLEX was able to start to solve the models, but still with a high gap even after 24 hours of computation. However, we observed that the gap returned by CPLEX as well as the gap to BKS are lower for LMVMPII in comparison with the values of LMVMP. The average gap returned by CPLEX with LMVMPII was around half of the average gap for LMVMP. The average gap to BKS was 2.19% for LMVMPII, while for LMVMP the gap was 4.96%.

The second observation is that for instances with 25 data centers and LMVMP model, CPLEX spent the whole time in the presolve phase without solving the root relaxation node. On the other hand, for LMVMPII, CPLEX was able to start a node exploration and found feasible solutions. For instances with 25 data centers, CPLEX explored an average of approximately 34800 nodes for instances with 100 VMs, 5700 nodes for instances with 150 VMs, and 1000 nodes for instances with 200 VMs. Even though CPLEX maintained a large gap, the gap with respect to the BKS was relatively small, considering the size of the instance and the difficulty to find feasible solutions (Stefanello et al., 2015a).

4.3. GRASP results

This subsection presents results and analyzes feasibility for different constructive heuristics combined with different local search procedures embedded in the GRASP framework. Also, we report experiments with the path-relinking procedure.

The main goals of the following experiments are show how difficult is to find feasible solutions using a constructive heuristic, and explore the effect of embedding a local search procedure to obtain a feasible solution. Finally, the last main objective is to evaluate the impact of each component added to the framework, i.e., constructive heuristic, local search method, and path-relinking strategy.

In the first part of our experiment, we evaluate the performance of each constructive heuristic described in Subsection 3.3, embedded in the GRASP. We performed one run for

Table 2: CPLEX detailed results for median and large instances.

		FO		CPLEX GAP		BKS GAP	
Instance	**BKS**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**
10_025_012_70	114,582.50	116,264.44	115,734.58	42.37	24.29	1.47	1.01
10_025_012_90	84,461.30	88,087.12	85,814.72	53.92	28.31	4.29	1.60
10_025_025_70	90,997.90	93,729.48	92,407.94	29.59	19.30	3.00	1.55
10_025_025_90	124,763.66	125,365.26	125,335.92	31.18	16.56	0.48	0.46
10_025_037_70	100,801.80	104,350.38	100,801.80	24.16	15.04	3.52	0.00
10_025_037_90	106,617.94	107,558.00	107,471.04	24.06	20.06	0.88	0.80
10_050_025_70	414,535.12	442,548.40	435,929.26	83.22	39.98	6.76	5.16
10_050_025_90	458,879.74	480,146.00	477,439.06	75.60	30.79	4.63	4.04
10_050_050_70	360,102.12	374,071.02	366,972.40	68.85	43.23	3.88	1.91
10_050_050_90	400,233.16	420,173.88	416,615.52	72.88	36.32	4.98	4.09
10_050_075_70	349,135.78	362,853.92	353,979.20	54.03	35.20	3.93	1.39
10_050_075_90	498,190.58	513,161.02	510,062.50	54.45	25.23	3.01	2.38
10_100_050_70	1,647,975.00	1,884,262.62	1,732,985.94	91.55	42.59	14.34	5.16
10_100_050_90	1,792,257.68	1,916,126.76	1,826,484.56	89.56	34.79	6.91	1.91
10_100_100_70	1,463,498.00	1,546,897.78	1,500,763.50	86.22	47.12	5.70	2.55
10_100_100_90	2,126,993.26	2,256,408.46	2,169,460.06	82.73	31.03	6.08	2.00
10_100_150_70	1,563,152.40	1,702,573.74	1,586,082.86	78.44	45.02	8.92	1.47
10_100_150_90	1,847,076.66	1,968,341.96	1,883,289.84	70.74	33.59	6.57	1.96
Average	-	-	-	61.86	31.58	4.96	2.19
25_100_050_70	1,887,688.24	-	1,976,649.48	-	44.15	-	4.71
25_100_050_90	2,116,849.00	-	2,184,190.20	-	34.89	-	3.18
25_100_100_70	1,953,155.20	-	2,056,428.04	-	44.55	-	5.29
25_100_100_90	2,021,228.76	-	2,091,723.56	-	35.72	-	3.49
25_100_150_70	1,967,364.52	-	2,061,181.36	-	43.91	-	4.77
25_100_150_90	2,160,014.54	-	2,235,936.02	-	35.35	-	3.51
25_150_075_70	4,603,163.50	-	4,768,514.02	-	42.99	-	3.59
25_150_075_90	4,618,491.80	-	4,786,179.50	-	36.54	-	3.63
25_150_150_70	3,882,650.94	-	4,085,463.10	-	46.34	-	5.22
25_150_150_90	4,706,129.66	-	4,899,667.66	-	36.82	-	4.11
25_150_225_70	4,340,090.18	-	4,530,866.46	-	46.00	-	4.40
25_150_225_90	4,523,393.44	-	4,708,994.42	-	35.51	-	4.10
25_200_100_70	6,937,008.98	-	7,539,073.36	-	50.79	-	8.68
25_200_100_90	9,034,147.98	-	9,391,760.06	-	34.93	-	3.96
25_200_200_70	7,146,330.32	-	7,579,453.48	-	49.13	-	6.06
25_200_200_90	8,578,620.94	-	8,909,349.84	-	36.53	-	3.86
25_200_300_70	7,638,447.16	-	8,033,640.92	-	42.32	-	5.17
25_200_300_90	8,195,152.30	-	8,579,652.20	-	38.94	-	4.69
Average	-	-	-	-	40.86	-	4.58

each instance, and for each combination of constructive heuristic and local search method. The stopping criterion of each run was a time limit of $|K| * |N| * \theta$ seconds, where $\theta = 0.8$. We use this stopping criterion in all experiments in order to provide a fair comparison for all evaluated strategies.

Table 3 shows the average of the percentage of feasible solutions found on each run of the algorithm for all instances and combinations of local search (rows on first column) and constructive heuristic (columns 2 to 5).

The first observation from Table 3 is that the probability of finding a feasible solution using only a constructive heuristic without local search is low (row `NoLS`). On average, the percentage of feasible solutions was less than 1%, with an average of less than 2% for the constructive heuristic *DC-Greedy*. However, the main observation is that the constructive

Table 3: Percentage of feasible solutions found by GRASP for different constructive heuristic and local search strategy.

LSType	*Random*	*Greedy*	*DC-Greedy*	*VM-Greedy*	**Average**
`NoLS`	0.00	1.02	1.96	0.57	0.89
`1V`	23.70	22.12	27.88	21.96	23.91
`2V`	77.26	78.02	79.72	78.41	78.35
`3V`	82.93	82.66	84.06	83.20	83.21
`4V`	84.08	83.61	84.98	84.19	84.22
Average	53.59	53.49	**55.72**	53.67	54.12

heuristic has a lower impact on the percentage of feasible solution in comparison with the impact of any local search method. When a local search procedure is considered (also used as repair procedure), the percentage of feasible solutions increases considerable. For example, using `2V` the percentage of feasible solutions is higher than 77% and higher than 82% for `3V`. This occurs even with the random constructive heuristic, showing that the local search has a performance that is not highly dependent of the initial solution. We also analyze the quality of the feasible solutions found with each strategy. The conclusions are similar to the percentage of feasible solutions, where the solution quality are similar for all constructive heuristic and tends to be better when we consider a more complex neighborhood search. Thus, in the remaining experiments we adopted the *DC-Greedy* constructive heuristic as default since it presents the highest percentage of feasible solutions.

Table 4 shows the percentage of feasible solutions for each instance for the constructive heuristic *DC-Greedy*.

Even with a significant increase in the percentage of feasible solutions found when the local search is applied, some instances still have a low percentage of feasible solutions. This shows that the set of instances used to evaluate the algorithms is diverse and not trivial to solve. We also observed that, as expected, most instances with a higher percentage of data center occupation have a low percentage of feasible solutions in comparison with the respective instance with a low percentage of occupation.

In the next set of experiments, the main objective is to evaluate the path-relinking component in the GRASP procedure. For each experiment, we performed five independent runs totalling an amount of 180 samples for each experiment. We also use $\theta = 0.8$ as stopping criterion The local search strategy used in the path-relinking is always the same one used after the construction phase in GRASP procedure. Experiments are chosen to analyze the performance for different combination of algorithms and to show the impact of each component.

To compare the results with respect to the best solution found on each run, we first scale the objective function value to the range $[0, 1]$ since each instance can have very different values. The scale is a simple transformation where for each instance, the largest cost over all analyzed experiments is scaled to 1 and the lowest is scaled to 0.

Table 4: Percentage of feasible solutions found by GRASP with the *DC-Greedy* constructive heuristic.

Instance	NoLS	1V	2V	3V	4V	Instance	NoLS	1V	2V	3V	4V
10_025_012_70	10.90	60.50	90.88	95.75	96.24	25_100_050_70	8.10	77.11	95.89	98.03	98.46
10_025_012_90	1.27	24.57	92.25	95.68	96.18	25_100_050_90	0.00	13.16	84.37	87.90	89.16
10_025_025_70	25.75	92.23	99.44	99.78	99.86	25_100_100_70	0.56	40.15	88.12	91.93	91.82
10_025_025_90	0.08	6.09	52.23	62.96	64.72	25_100_100_90	0.00	3.08	66.90	74.69	75.48
10_025_037_70	2.91	59.80	81.60	86.55	88.67	25_100_150_70	1.23	46.98	93.80	96.42	97.31
10_025_037_90	0.06	4.93	44.24	57.58	63.35	25_100_150_90	0.00	2.78	66.30	73.97	75.14
10_050_025_70	4.38	42.79	94.45	95.80	96.17	25_150_075_70	5.84	60.39	95.35	96.03	97.12
10_050_025_90	0.13	21.10	88.80	92.63	93.11	25_150_075_90	0.01	6.60	84.39	87.73	88.13
10_050_050_70	0.34	23.77	71.09	76.97	78.78	25_150_150_70	0.77	31.90	79.68	84.00	85.29
10_050_050_90	0.05	13.24	56.53	65.21	65.81	25_150_150_90	0.00	5.23	64.75	70.49	69.27
10_050_075_70	1.50	49.55	72.63	77.26	78.96	25_150_225_70	0.53	31.81	80.36	83.15	83.33
10_050_075_90	0.07	10.02	68.59	72.29	74.02	25_150_225_90	0.00	1.59	47.46	57.90	56.01
10_100_050_70	3.34	65.63	98.49	98.96	98.62	25_200_100_70	0.21	31.49	85.27	86.11	90.86
10_100_050_90	0.01	3.07	80.60	86.54	86.99	25_200_100_90	0.00	21.90	95.36	97.76	97.36
10_100_100_70	1.66	39.19	87.45	91.28	93.38	25_200_200_70	0.07	22.82	75.00	80.57	81.38
10_100_100_90	0.00	7.45	84.02	86.81	86.66	25_200_200_90	0.00	21.36	93.51	95.41	95.36
10_100_150_70	0.82	41.94	83.35	86.01	86.89	25_200_300_70	0.00	13.36	66.92	72.53	73.55
10_100_150_90	0.00	1.46	73.33	74.18	75.63	25_200_300_90	0.00	4.49	86.54	89.12	90.08
Average	2.96	31.52	78.89	83.46	84.67	Average	0.96	24.23	80.55	84.65	85.28

Figure 3 shows the distribution of the scaled cost for each algorithm. The box plots show the location of the minimum value, lower quartile, median, upper quartile, and maximum value of each experiment. The dots are the outliers. Experiments are represented on the horizontal axis and are labelled as composition of main algorithm name, path-relinking type, local search strategy, and size of elite set parameter. Experiments ended with "$*$" use a full exploration on the neighborhood *chain3L*, i.e, $\alpha_{3L} = 1$ and $\beta_{3L} = |N|$. Also, note that the first three experiments are a simple GRASP heuristic and do not include a path-relinking component.

The first observation regards the relation between the intensification of the neighborhood exploration and restarting the search. On the one hand, from Table 3 and 4 we observe that the local search that includes more complex neighborhoods overcomes in most cases the searches with less complex neighborhoods in the percentage of feasible solutions. Since the strategies are executed for the same time, is natural that a less complex search has a higher number of iterations and a higher numbers of restarts. This is an indication that for this problem it is preferable to invest in the exploration of the neighborhood of a high-quality solution rather than making a full restart to a new point of the search space. On the other hand, even with a higher ability of the local search 4V in producing feasible solutions and exploring a large neighborhood, when we consider the best solution found in the experiments, local search 3V produces better results than 4V. This indicates that for the proposed neighborhood strategies, there is a trade-off between intensification of the neighborhood exploration and time spent on each local search strategy. This can be used to guide the search and select the best strategies for a general proposed method.

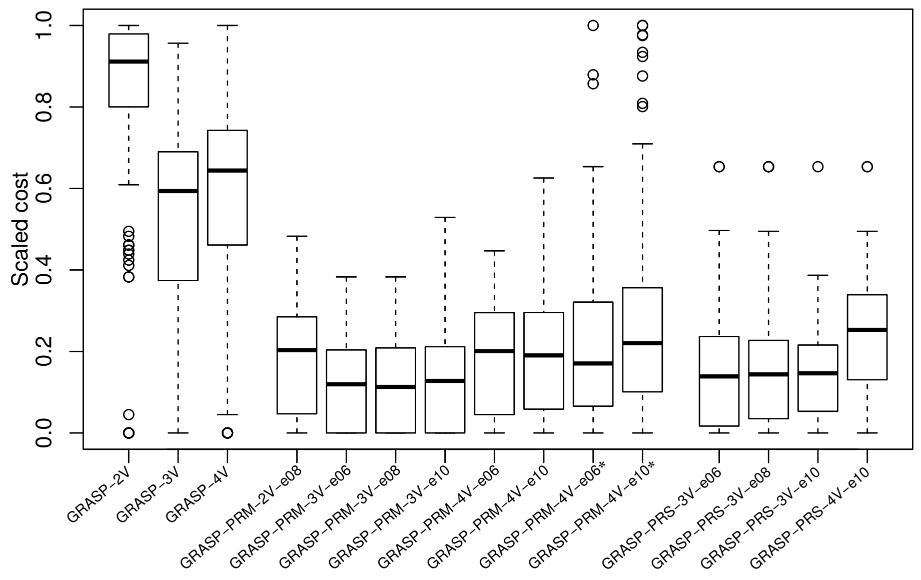


Figure 3: Dispersion of scaled cost for each algorithm.

The second observation is that the path-relinking component improves the results significantly in comparison with the case without this strategy (case GRASP-2V, GRASP-3V, and GRASP-4V). We also observe that both path-relinking strategies have similar performance. Furthermore, the results are not influenced heavily by the values of elite size in the tested interval.

To confirm the results presented in Figure 3, we use the R package to test the normality of these distributions using the Shapiro-Wilk test and apply the Mann-Whitney-Wilcoxon U test. For all tests, we assume a confidence interval of 99%. Shapiro-Wilk tests indicate that no cost distribution fits a normal distribution since the p-values for all tests are less than 0.01. Therefore, we applied the U test which assumes as null hypothesis that the location statistics are equal in both distributions. We also use a p-value correction procedure based on false discovery rate (FDR) to minimize the number of false positives.

Table 5 shows U test results for each pair of algorithms. The structure of this table is as follows: Each row and column is indexed by one algorithm. Each element in the diagonal (bold) is the median of the scaled cost of the corresponding algorithm. The upper-right diagonal elements are the differences in location statistics for each pair of algorithms. A negative difference indicates that the "row algorithm" has its location statistics lower (better) than the "column algorithm", and the positive difference is the opposite. The bottom-left diagonal elements are the p-values of each test. Math signals indicate when $p<0.01$ for a U test between "row algorithm" and "column algorithm" for the respective signal alternative hypothesis. The case "less" indicates that the "row algorithm" overcomes the "column algorithm", or the opposite in the case "greater".

Table 5 confirms that the difference between the results of the standard GRASP procedure and the procedure that includes the path-relinking component are statistically signifi-

Table 5: Values of medians, p-values, and difference in median location for cost distributions using a confidence interval of 99% for GRASP algorithm.

	GRASP			**GRASP-PRM**								**GRASP-PRS**			
	2V	3V	4V	2V-e08	3V-e06	3V-e08	3V-e10	4V-e06	4V-e10	5V-e06	5V-e10	3V-e06	3V-e08	3V-e10	4V-e10
2V	**0.911**	0.311	0.265	0.683	0.755	0.756	0.751	0.683	0.679	0.676	0.642	0.730	0.728	0.734	0.647
3V	$<$	**0.593**	-0.048	0.379	0.452	0.456	0.448	0.387	0.379	0.360	0.332	0.424	0.422	0.427	0.342
4V	$<$	$>$	**0.644**	0.426	0.496	0.498	0.489	0.434	0.421	0.416	0.389	0.468	0.465	0.476	0.393
PRM-2V-e08	$<$	$<$	$<$	**0.203**	0.075	0.081	0.069	-0.001	0.003	-0.014	-0.051	0.056	0.053	0.061	-0.035
PRM-3V-e06	$<$	$<$	$<$	$<$	**0.119**	0.002	-0.006	-0.077	-0.075	-0.085	-0.121	-0.022	-0.029	-0.017	-0.118
PRM-3V-e08	$<$	$<$	$<$	$<$	0.839	**0.113**	-0.010	-0.077	-0.082	-0.086	-0.125	-0.027	-0.028	-0.022	-0.119
PRM-3V-e10	$<$	$<$	$<$	$<$	0.568	0.353	**0.128**	-0.072	-0.075	-0.082	-0.116	-0.020	-0.017	-0.012	-0.110
PRM-4V-e06	$<$	$<$	$<$	0.930	$>$	$>$	$>$	**0.201**	-0.007	-0.014	-0.049	0.051	0.054	0.058	-0.039
PRM-4V-e10	$<$	$<$	$<$	0.846	$>$	$>$	$>$	0.582	**0.190**	-0.015	-0.045	0.056	0.055	0.058	-0.037
PRM-5V-e06	$<$	$<$	$<$	0.409	$>$	$>$	$>$	0.376	0.294	**0.171**	-0.041	0.063	0.058	0.067	-0.031
PRM-5V-e10	$<$	$<$	$<$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	**0.220**	0.100	0.093	0.096	0.006
PRS-3V-e06	$<$	$<$	$<$	$<$	0.075	0.040	0.089	$<$	$<$	$<$	$<$	**0.139**	-0.004	0.003	-0.097
PRS-3V-e08	$<$	$<$	$<$	$<$	0.016	0.010	0.091	$<$	$<$	$<$	$<$	0.737	**0.144**	0.009	-0.097
PRS-3V-e10	$<$	$<$	$<$	$<$	0.065	0.066	0.243	$<$	$<$	$<$	$<$	0.774	0.384	**0.146**	-0.102
PRS-4V-e10	$<$	$<$	$<$	$>$	$>$	$>$	$>$	$>$	$>$	0.056	0.774	$>$	$>$	$>$	**0.253**

cant for a confidence interval of 99%. Tests between cases with local search `3V` and different elite sizes indicate that the difference is not significant, even for both path-relinking strategies. However analyzing the values of median and median location, we conclude that there is a low advantage for the `PRM` strategy with elite size equal to 8.

A report for the solution quality obtained with this strategy is shown in Subsection 4.5. Next we report results for the BRKGA.

4.4. BRKGA results

This subsection presents results for the biased random-key genetic algorithm. The objective is to analyze different procedures, decoders, the recode process, and the path-relinking procedure introduced in the algorithm. The experiments follow the rules and the objectives explained in the previous subsection.

BRKGA parameters was set to the same parameters as described in Stefanello et al. (2015a) which are similar to the values suggested by Gonçalves and Resende (2011), i.e., the elite size $p_e = 0.24p$, the set of mutants $p_m = 0.12p$, and the probability of inheriting $\rho_A = 0.6$. The restart parameter was disabled and the population size $p = 75$. The stopping criterion for all runs was a time set to $|K| * |N| * \theta$ seconds (where $\theta = 0.8$).

We run experiments that include different local searches, decoders and path-relinking procedures. In each experiment, we performed five independent runs with different random seeds for each instance, given a total of 180 runs. To compare the results we scale the values of the objective function for the best solution found on each run to the range $[0, 1]$, as described in the previous subsection.

Figure 4 shows the box plot for the distribution of the scaled cost for each algorithm. Algorithms are represented on the horizontal axis and are labelled as a composition of main algorithm name, path-relinking strategy, decoder method, local search strategy, and size of elite set parameter.

The first observation is that the recode process helps to improve the results, since all experiments with decoder D3 have medians lower than the respective corresponding algorithm with decoders D1 and D2. Note for example that without the recode, one modification

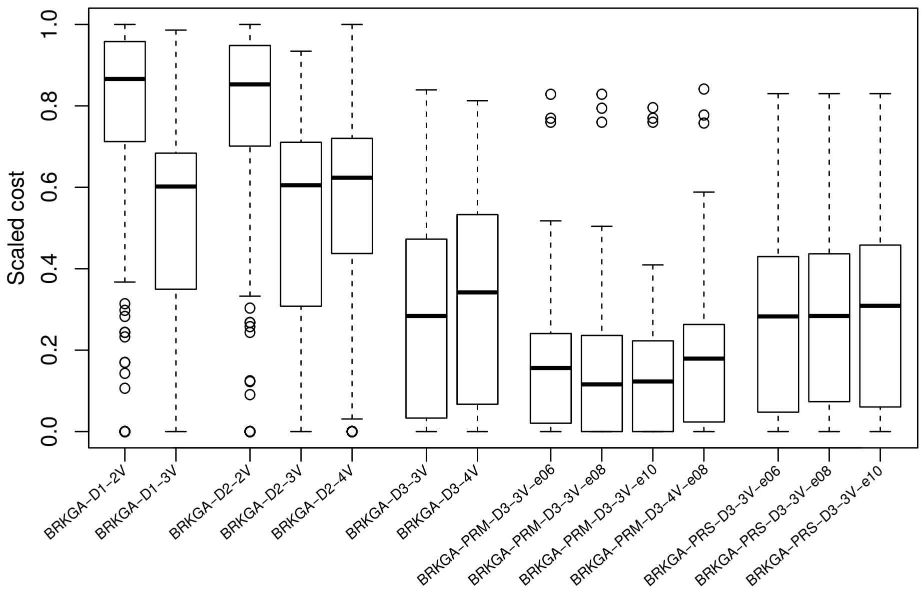


Figure 4: Dispersion of scaled cost for each algorithm.

in a random key can have influence in one or more placements (since the local search can apply many changes). Using the recode, when one key is changed, the modification is in only one placement. This well-defined behavior helps to keep a more accurate information of each individual. The second observation is that for the local search strategies, the same conclusions from the GRASP can be taken for BRKGA. Experiments with local search `3V` provides better results than the other local search strategies. Finally, strategies that include the path-relinking procedure are more effective to producing better results than the standard counterpart. However, different from what was observed with GRASP, the difference between `PRM` and `PRS` is significant. We attribute the worst performance of `PRS` due to the small number of applications of the path-relinking since it is limited to only a few generations. Also, after some iterations of BRKGA, the solutions in the elite set of BRKGA tend to be similar, and fail in the similarity check with the solution in the elite set of path-relinking, and thus, the path-relinking operator is not performed.

Table 6 shows U test results for each pair of algorithms. We present the values of medians, p-values, and the difference in median location for the scaled cost distributions using a confidence interval of 99% for all experiments. The organization of this table follows the description of Table 5.

Table 6 confirms the considerations described above showing that the best approaches use the path-relinking component (`PRM`). Also, as mentioned for GRASP in the previous subsection, the experiments for different elite sizes indicate that the differences are not statistically significant for a confidence interval of 99%. However, analyzing the values of the median, and median location we conclude that with a elite set of size 8, the procedure tends to produce better results.

Table 6: Values of medians, p-values, and difference in median location for cost distributions using a confidence interval of 99% for BRKGA algorithm.

	BRKGA							**BRKGA-PRM**				**BRKGA-PRS**		
	D1-2V	D1-3V	D2-2V	D2-3V	D2-4V	D3-3V	D3-4V	3V-e06	3V-e08	3V-e10	4V-e08	3V-e06	3V-e08	3V-e10
D1-2V	**0.866**	0.251	0.000	0.242	0.211	0.512	0.449	0.678	0.697	0.701	0.645	0.533	0.520	0.515
D1-3V	<	**0.602**	-0.247	0.000	-0.028	0.250	0.185	0.413	0.437	0.440	0.397	0.275	0.263	0.261
D2-2V	0.987	>	**0.853**	0.243	0.200	0.505	0.440	0.669	0.691	0.692	0.640	0.524	0.512	0.506
D2-3V	<	0.979	<	**0.605**	-0.036	0.250	0.188	0.424	0.443	0.449	0.397	0.272	0.265	0.260
D2-4V	<	0.016	<	>	**0.623**	0.275	0.216	0.449	0.470	0.477	0.424	0.300	0.291	0.287
D3-3V	<	<	<	<	<	**0.284**	-0.062	0.179	0.208	0.202	0.142	0.031	0.014	0.018
D3-4V	<	<	<	<	<	>	**0.342**	0.233	0.266	0.263	0.206	0.091	0.073	0.075
PRM-3V-e06	<	<	<	<	<	<	<	**0.156**	0.020	0.023	-0.026	-0.151	-0.163	-0.169
PRM-3V-e08	<	<	<	<	<	<	<	0.074	**0.116**	0.001	-0.042	-0.176	-0.186	-0.190
PRM-3V-e10	<	<	<	<	<	<	<	0.040	0.891	**0.123**	-0.046	-0.179	-0.194	-0.193
PRM-4V-e08	<	<	<	<	<	<	<	0.047	>	>	**0.179**	-0.118	-0.129	-0.133
PRS-3V-e06	<	<	<	<	<	0.011	<	>	>	>	>	**0.283**	-0.015	-0.015
PRS-3V-e08	<	<	<	<	<	0.171	<	>	>	>	>	0.102	**0.284**	-0.003
PRS-3V-e10	<	<	<	<	<	0.071	<	>	>	>	>	0.118	0.810	**0.309**

In Stefanello et al. (2015a) this problem was first introduced, and the first version of BRKGA is presented. From the original approach, three main improvements are made. First, we added a new neighborhood search. In the original approach, we described a local search that includes only shift and swap moves (namely LSW that correspond to `2V` in this paper). We observe from the results in Figure 4 and in Table 6 that the inclusion of this new neighborhood strategy (*chain search*) helps to significantly improve the results (comparison between BRKGA-D2-2V and BRKGA-D2-3V). Second, the inclusion of the recode procedure in the decoder D2 (defined as decoder D3), also significantly improves the results (comparison between BRKGA-D2-2V and BRKGA-D3-2V, as well as BRKGA-D2-3V and BRKGA-D3-3V). Finally, we included a path-relinking component that, to the best of our knowledge, is the first experiment that considers this hybrid approach between BRKGA and path-relinking.

A final observation about running times of BRKGA is that results are reported using single-thread processing to provide a fair comparison with the GRASP and CPLEX. However, the BRKGA API of Toso and Resende (2015) provides for efficient multi-thread decoding that could be used to substantially reduce the running times when multiple processors are available.

4.5. Additional comparison

In this subsection we report an overview over the best strategies for BRKGA and GRASP, providing addition comparisons and information for both algorithms. Figure 5 shows an overview for the experiments for both strategies reported in the previous subsections.

As depicted in Figure 5 and reported in Subsections 4.3 and 4.4, the best approach for both algorithms is consider the local search `3V` and to inclusion of the path-relinking component with elite size equal to 8. Table 7 presents supplementary information for the experiments with both strategies. The second column shows the best-known solution value (BKS) for each instance. The next columns show the minimum, average and maximum percentage gap for both algorithms. The percentage gap is calculated by the formula (FO-BKS)*100/BKS, where FO is the corresponding value of the objective function for the

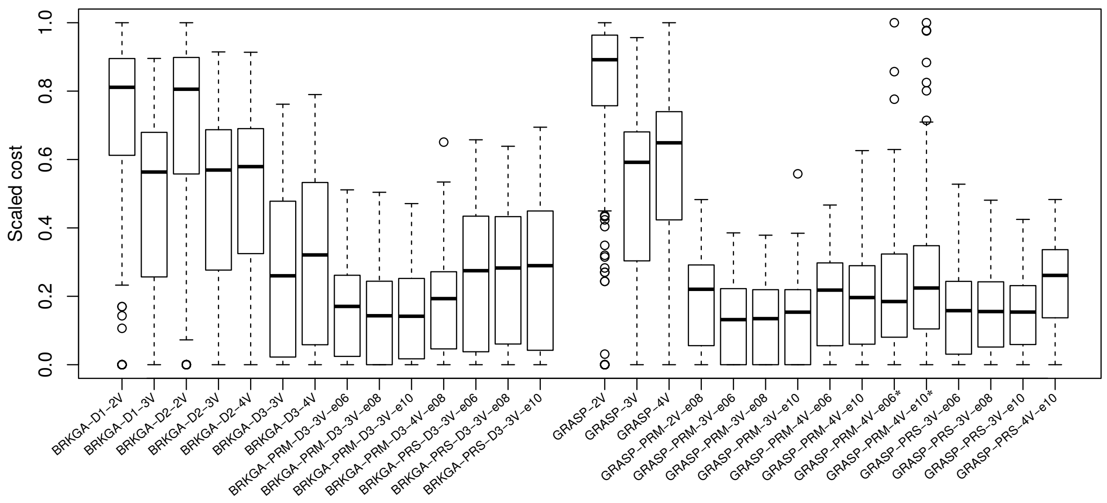


Figure 5: Dispersion of scaled cost for each algorithm.

respective algorithm. Finally, columns Time shows the minimum and average time in seconds in which each algorithm obtained the last improvement in the objective function.

From this table, we observe that the percentage gap is relatively small for both algorithms, showing that the approaches are efficient in finding good quality solutions. For instances with 10 data centers, in most cases both algorithms found the BKS. Also, considering that each run was stopped by the time limit ($|N| * |K| * 0.8$), from columns Time we observe that a significant portion of the execution was spent without improvements. This indicates that the algorithm has a fast convergence, and will probably provide a good solution if stopped before the used time limit. Furthermore, we observe that BRKGA-PR is slightly faster than GRASP-PR for the instance set with 10 data centers. However, the opposite happens for the instances with 25 data centers.

Finally, the last experiment uses the Time-To-Target (TTT) plots to display the running time distribution for the algorithm to find a solution at least as good as a given target value. TTT plots were proposed by Feo et al. (1994); Aiex et al. (2007) and have been advocated by Hoos and Stützle (1998b;a) as a way to characterize the running times of stochastic algorithms for optimization problems. The experiment consists in performing 200 runs of BRKGA-PR and GRASP-PR algorithms for two instances until a target is reached. The target was set to be the worst solution found for both strategies in the previous experiment, i.e, 1,468,973 for `010_100_100_70` and 9,077,672 for `025_200_100_90`. We choose one instance with 10 and one instance with 25 data centers, both with the lowest difference of average percentage gap between both approaches (excluding equal values).

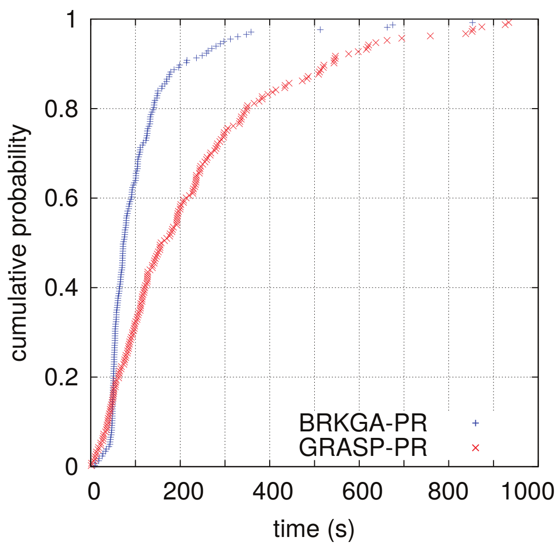
Figure 6 illustrates the cumulative probability plot obtained by using BRKGA-PR and GRASP-PR for two instances. For instance `10_100_100_70`, we observe that BRKGA-PR has a higher probability of finding a solution at least as good as the target in less time than GRASP-PR. For instance `25_150_225_70` likelihood is reverse.

In summary, both approaches have good performance according to specific scenarios. The best performance of BRKGA-PR is on small and median size instances while GRASP-

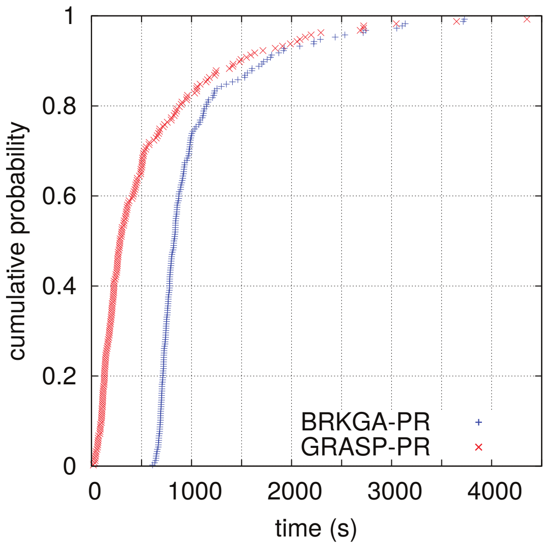
Table 7: Comparison of percentage gap and last improve time for BRKGA-PR and GRASP-PR.

		BRKGA-PR					**GRASP-PR**				
		GAP (%)			**Time (s)**		**GAP (%)**			**Time (s)**	
Instance	**BKS**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Min**	**Avg**	**Max**	**Min**	**Avg**
10_025_012_70	114,582.50	0.00	0.00	0.00	1.6	22.7	0.00	0.00	0.00	1.3	38.7
10_025_012_90	84,461.30	0.00	0.32	0.54	1.8	35.9	0.00	0.00	0.00	16.1	49.2
10_025_025_70	90,997.90	0.00	0.00	0.00	0.5	0.8	0.00	0.00	0.00	48.5	111.3
10_025_025_90	124,763.66	0.00	0.15	0.18	1.9	45.1	0.00	0.04	0.18	19.1	59.8
10_025_037_70	100,801.80	0.00	0.00	0.00	0.7	0.8	0.00	0.00	0.00	0.4	1.0
10_025_037_90	106,617.94	0.00	0.00	0.00	0.7	13.3	0.00	0.00	0.00	1.1	2.3
10_050_025_70	414,535.12	0.11	0.26	0.47	12.5	86.5	0.00	0.08	0.13	47.9	190.9
10_050_025_90	458,879.74	0.00	0.02	0.07	38.3	76.0	0.00	0.09	0.16	87.5	193.6
10_050_050_70	360,102.12	0.00	0.10	0.49	11.4	36.0	0.00	0.00	0.00	4.5	92.4
10_050_050_90	400,233.16	0.00	0.37	0.64	21.2	32.9	0.00	0.28	0.62	32.9	185.6
10_050_075_70	349,135.78	0.00	0.00	0.00	4.7	29.8	0.00	0.00	0.00	7.7	62.1
10_050_075_90	498,190.58	0.39	0.58	0.87	47.9	153.7	0.00	0.26	0.47	18.1	147.8
10_100_050_70	1,647,975.00	0.00	1.98	3.32	82.8	395.4	1.12	2.26	3.37	457.5	652.2
10_100_050_90	1,792,257.68	0.01	0.18	0.31	98.1	380.7	0.14	0.31	0.55	292.5	493.0
10_100_100_70	1,463,498.00	0.00	0.08	0.37	50.9	271.7	0.00	0.03	0.14	518.3	624.6
10_100_100_90	2,126,993.26	0.00	0.23	0.48	433.7	656.5	0.12	0.28	0.53	60.6	483.5
10_100_150_70	1,563,152.40	0.00	0.13	0.40	190.1	437.5	0.06	0.30	0.59	198.7	537.4
10_100_150_90	1,847,076.66	0.01	0.12	0.32	137.2	429.0	0.08	0.26	0.39	252.6	575.8
Average		0.03	0.25	0.47	63.1	172.5	0.08	0.23	0.40	114.7	250.0
25_100_050_70	1,887,688.24	0.18	0.43	0.78	160.3	1,128.9	0.12	0.30	0.47	953.2	1,464.6
25_100_050_90	2,116,849.00	0.03	0.52	0.81	904.3	1,615.4	0.26	0.37	0.46	741.2	1,326.4
25_100_100_70	1,953,155.20	0.36	0.62	0.83	284.5	770.2	0.00	0.35	0.75	253.9	1,122.9
25_100_100_90	2,021,228.76	0.35	0.72	1.24	976.0	1,516.7	0.44	0.55	0.65	847.5	1,207.5
25_100_150_70	1,967,364.52	0.30	0.57	0.82	276.5	1,141.6	0.00	0.42	0.70	160.5	1,377.1
25_100_150_90	2,160,014.54	0.56	0.80	1.08	776.3	1,509.1	0.44	0.62	0.77	528.9	1,156.7
25_150_075_70	4,603,163.50	0.20	0.37	0.49	531.1	1,877.3	0.24	0.33	0.41	992.6	1,567.2
25_150_075_90	4,618,491.80	0.21	0.33	0.49	1,223.7	1,807.9	0.21	0.31	0.40	1,986.0	2,548.8
25_150_150_70	3,882,650.94	0.19	0.36	0.54	788.5	2,183.6	0.19	0.34	0.49	1,560.7	2,156.1
25_150_150_90	4,706,129.66	0.41	0.53	0.68	597.5	1,984.0	0.32	0.46	0.58	889.2	2,067.9
25_150_225_70	4,340,090.18	0.13	0.34	0.53	1,657.6	2,347.1	0.47	0.59	0.80	1,648.5	2,321.8
25_150_225_90	4,523,393.44	0.44	0.54	0.69	271.5	1,684.5	0.17	0.37	0.53	543.2	1,607.5
25_200_100_70	6,937,008.98	0.22	0.28	0.35	3,123.8	3,938.7	0.07	0.26	0.44	1,855.8	2,841.7
25_200_100_90	9,034,147.98	0.15	0.29	0.48	2,017.7	3,352.2	0.15	0.29	0.44	1,371.5	2,688.1
25_200_200_70	7,146,330.32	0.13	0.24	0.44	1,271.1	2,948.6	0.10	0.35	0.52	1,718.1	3,201.1
25_200_200_90	8,578,620.94	0.00	0.15	0.26	1,170.0	2,669.4	0.18	0.22	0.30	326.3	1,681.0
25_200_300_70	7,638,447.16	0.00	0.25	0.35	1,763.6	3,216.9	0.19	0.28	0.39	819.1	2,745.9
25_200_300_90	8,195,152.30	0.05	0.18	0.27	1,523.7	2,689.1	0.13	0.27	0.37	913.1	2,796.6
Average		0.22	0.42	0.62	1,073.2	2,132.3	0.20	0.37	0.53	1,006.1	1,993.3

PR has a better performance on large instances. The path-relinking component provides an improvement for both strategies. We attribute this performance to a rugged landscape for this problem and the fact that is hard to find feasible solutions. Since in general path-relinking is performed between two feasible solutions, this component explores a search space where solutions are feasible or has fewer infeasibilities. Thus, when local search is applied



(a) TTT plot for 10_100_100_70.



(b) TTT plot for 25_200_100_90.

Figure 6: Cumulative probability distribution.

to a solution in the path, the search tends to be faster and leads to a new local minimum that can be better than the initial and guiding solutions.

Finally, we evaluate our algorithm considering a set of instances of sizes not address before in the literature. We test for instances with up to 25 data centers and 200 virtual machines. Even though in real world application 200 VMs can be considered small-scale, previous work on related problems addressed instances with up to 50. A major contribution of our work is to consider five times larger than those previously considered. Furthermore, we provide a new set of benchmark instances for a future experimental analyses of algorithms.

In the next subsection we evaluate our proposed algorithms on an adapted set of instances from a similar problem and compare our algorithm with a method described in the literature.

4.6. Results for the Generalized Quadratic Assignment Problem

VMPlacement is a generalization of GQAP and our proposed method can be easily adapted for GQAP. Since we evaluate the infeasibilities in the objective function, we only need to change the cost evaluation function. However, for convenience, we adapted the instances of GQAP to make them instances of the VMPlacement. To do this, we give a sufficiently large value for bandwidth capacity between each pair of data centers as well as for the required latency between each pair of virtual machines. Finally, we also define an empty set of users.

The main objective of this experiment is to provide a brief comparison with a method described in the literature to show that our approach has a competitive performance. We first perform an experiment with CPLEX to provide a baseline comparison with exact methods. Later, we compare our algorithms with a GRASP with path-relinking proposed by Mateus et al. (2010).

We use a set of instances proposed by Cordeau et al. (2006). These instances have 20 to 50 facilities (virtual machines) and 6 to 20 locations (data centers). Instances are listed in Table 8 and are labelled, respectively, with the number of facilities (virtual machines), locations (data centers), and a parameter that controls the tightness of the capacity constraints. The higher the value of this parameter, the higher the tightness of the capacity constraints. Since this set comprises the largest instance set available for GQAP, we evaluate our strategies only for this set of instances. These experiments are carried out on a computer with an Intel(R) Core(TM) i7 CPU 930 with 2.80 GHz and 12GB of main memory.

Cordeau et al. (2006) evaluate CPLEX (version 8.1) exploring the emphasis parameter on the branch-and-bound tree for a time limit of two hours. CPLEX was able to prove optimality only for instance `30-08-55`. Pessoa et al. (2010) proposed exact algorithms that combine branch-and-bound with a new Lagrangean decomposition and the Reformulation-Linearization Technique and prove optimality for another 13 instances of this instance set.

In the first experiment we evaluate the performance of CPLEX on models LMVMP and LMVMPII, described in Section 2. We run CPLEX for each instance with a time limit of one day (86,400 seconds). We also set the tree memory parameter (`TreLim`) to 5000 to stop execution when this memory limit is exceeded. The remaining parameters are kept to their default values.

Table 8 shows the results for each instance listed in the first column. The second column shows the best-known solutions for each instance (optimal values are given in boldface). The next two sets of columns show the results for each linear mathematical model. Column Nodes shows the number of search tree nodes visited by CPLEX. Column Integer Sol shows the objective function of best solution found during the execution. Column gap shows the percentage gap between the lower bound and the best integer solution found by CPLEX. Column Time shows the running time in seconds for CPLEX to prove optimality or satisfy the stopping criterion. Instances that CPLEX found the optimal solution or proved optimality are highlighted in boldface.

Observing the results in Pessoa et al. (2010) and comparing then with the results in Table 8, we note that the branch-and-bound approach in Pessoa et al. (2010) tends to be more efficient than CPLEX. In general, the running time tends to be smaller than the required by CPLEX. However, both methods are not strictly comparable since the experiments use different stopping criteria, and are reported over different computers. In the end, the main observation from these results is that even with a high increase in the computational power and the development of many algorithms and improvements over each CPLEX version, this set of instances is still hard to be solved by exact methods.

In a second experiment, we compare our strategies with the GRASP-PR proposed by Mateus et al. (2010), the state-of-the-art heuristic for GQAP. For each instance we performed 200 independent runs. Each run stopped when a solution value as good as the best-known solution was found (column BKS in Table 8).

Two main differences are observed between our approaches and GRASP-PR proposed by Mateus et al. (2010). First, we performed a wide exploration with the local search strategy. Mateus et al. (2010) uses an approximate local search using shift and swap moves. In our approach, we explore all neighbors using three neighborhood structures. Second, we

Table 8: CPLEX results for GQAP instances.

		LMVMP				LMVMPII			
Instance	**BKS**	**Nodes**	**Integer Sol**	**gap** (%)	**Time** (s)	**Nodes**	**Integer Sol**	**gap** (%)	**Time** (s)
20-15-35	**1,471,896**	2,990	**1,471,896**	**0.00**	3,788.3	30,709,073	**1,471,896**	7.95	41,981.1
20-15-55	**1,723,638**	10,839	**1,723,638**	**0.00**	9,149.2	30,269,870	**1,723,638**	6.58	39,265.6
20-15-75	**1,953,188**	5,725	**1,953,188**	**0.00**	4,546.3	1,867,883	**1,953,188**	**0.00**	3,979.6
30-06-95	**5,160,920**	24,889	**5,160,920**	**0.00**	9,471.3	16,820,694	**5,160,920**	**0.00**	35,086.4
30-07-75	**4,383,923**	47,884	**4,383,923**	**0.00**	22,429.5	26,644,434	**4,383,923**	4.50	63,524.3
30-08-55	**3,501,695**	793	**3,501,695**	**0.00**	370.1	14,090,023	**3,501,695**	**0.00**	36,009.5
30-10-65	**3,620,959**	41,020	**3,620,959**	**0.00**	65,805.4	19,972,814	**3,620,959**	13.32	58,100.3
30-20-35	**3,379,359**	1,872	3,605,129	23.96	86,400.0	14,857,869	**3,379,359**	35.36	50,023.8
30-20-55	**3,593,105**	1,680	3,865,716	31.23	86,400.0	14,117,019	**3,593,105**	30.27	54,253.0
30-20-75	**4,050,938**	1,946	4,245,753	26.43	86,400.0	13,553,603	**4,050,938**	21.13	59,363.0
30-20-95	**5,710,645**	3,575	5,840,934	12.96	86,400.0	6,869,516	**5,710,645**	2.71	86,400.2
35-15-35	**4,456,670**	3,290	**4,456,670**	13.23	86,400.0	15,797,188	4,457,348	34.76	40,190.5
35-15-55	**4,639,128**	3,078	4,723,959	19.34	86,400.0	13,903,550	**4,639,128**	27.52	40,295.3
35-15-75	6,301,723	1,139	6,395,402	27.83	86,400.0	13,348,410	6,327,723	28.02	59,428.9
35-15-95	**6,670,264**	1,546	7,370,866	32.09	86,400.0	9,664,857	6,689,421	20.38	86,400.0
40-07-75	7,405,793	34,769	**7,405,793**	3.60	86,400.0	20,466,362	**7,405,793**	12.94	35,228.8
40-09-95	7,667,719	8,471	7,941,330	18.26	86,400.0	16,718,721	7,694,904	18.61	38,337.7
40-10-65	7,265,559	6,790	7,305,381	10.80	86,400.0	16,475,064	**7,265,559**	22.79	49,526.1
50-10-65	10,513,029	5,467	**10,513,029**	4.15	86,400.0	14,617,831	**10,513,029**	19.39	52,238.3
50-10-75	11,217,503	2,209	11,415,840	19.89	86,400.0	14,018,304	11,251,072	24.42	54,332.5
50-10-95	12,845,598	1,960	13,242,115	18.57	86,400.0	13,205,512	**12,845,598**	19.25	57,175.5

use a penalization strategy to deal with infeasible solutions. This allows visiting infeasible solutions to reach new feasible solutions that can be difficult to be reached only exploring a feasible search space.

Table 9 shows a comparison of GRASP-PR of Mateus et al. (2010) and our best two approaches described in the previous subsection. Columns Min, Avg, Max, Sd give the minimum, average, and maximum times, as well as the standard deviation of these runs to find a solution with values equal to BKS. Finally, column 0.95 shows the time in which 95% of the runs find the BKS.

The results described in Mateus et al. (2010) are reported on a Dell PE1950 computer with dual quad core 2.66 GHz Intel Xeon processors. Unfortunately, there is no specific information about the processor model. We assume the model Intel(R) Xeon(TM) X5355 2.66 GHz, based on server model, the number of cores and clock frequency. We run our algorithms for this set of experiments on a computer with an Intel(R) Core(TM) i7 CPU 930 2.80 GHz. The machine that we use is approximately 1.2 times faster than the one used in Mateus et al. (2010) (based on Single Thread Rating data from `wwww.cpubenchmark.net`). The execution times for the two implementations are not strictly comparable since the languages and compilers used are different. However, at a high level, this comparison provides an idea of the behavior of the algorithms.

Even considering that the running times are measured on different hardware, the data from Table 9 shows that our version of GRASP-PR, as well as the BRKGA-PR approach have a significant lower running time in comparison with the results reported in Mateus et al. (2010). The average minimum time is nearly two orders of magnitude smaller. This

Table 9: Comparison algorithms for GQAP.

	Mateus et al. (2010)					**BRKGA-PR**					**GRASP-PR**				
Instance	Min	Avg	Max	Sd	0.95	Min	Avg	Max	Sd	0.95	Min	Avg	Max	Sd	0.95
20-15-35	0.16	7.05	38.87	6.47	21.04	0.06	0.13	0.28	0.04	0.17	0.01	0.08	0.57	0.09	0.22
20-15-55	0.24	2.87	14.42	2.18	7.69	0.06	0.12	0.17	0.03	0.16	0.01	0.13	0.75	0.14	0.36
20-15-75	0.26	2.01	12.82	1.72	5.25	0.08	0.39	3.44	0.50	1.32	0.01	0.41	3.30	0.51	1.41
30-06-95	0.55	2.59	23.81	2.22	6.44	0.16	0.41	2.23	0.25	0.88	0.01	0.48	1.78	0.36	1.16
30-07-75	0.50	7.80	38.47	5.47	18.18	0.15	0.66	5.22	0.63	1.86	0.01	0.48	4.10	0.49	1.28
30-08-55	0.18	1.61	4.89	0.95	3.60	0.13	0.27	0.40	0.06	0.35	0.01	0.03	0.24	0.03	0.10
30-10-65	2.75	121.94	1,032.80	146.06	514.82	0.18	1.58	12.27	1.83	5.35	0.01	1.59	8.83	1.53	4.41
30-20-35	1.08	79.03	4,441.40	312.62	166.21	0.24	0.73	1.73	0.29	1.27	0.01	1.81	8.60	1.60	5.14
30-20-55	1.28	25.16	150.11	21.19	66.82	0.25	1.35	9.84	1.37	4.05	0.01	1.73	7.43	1.37	4.58
30-20-75	2.11	41.43	759.81	68.39	148.43	0.29	0.71	1.47	0.23	1.15	0.01	1.01	4.84	0.90	2.67
30-20-95	833.99	543,019.01	2,533,608.00	747,962.39	2,186,440.80	8.82	514.61	2,780.71	491.24	1,556.91	1.40	1,880.75	7,363.47	1,643.47	5,563.42
35-15-35	8.41	306.11	1,717.94	242.49	775.25	0.24	5.62	27.82	5.64	17.71	0.09	12.26	53.11	11.26	35.16
35-15-55	4.33	21.13	75.69	11.95	42.47	0.32	0.97	4.10	0.54	1.71	0.01	0.38	1.56	0.31	0.97
35-15-75	5.18	68.23	621.83	74.17	183.19	0.35	1.09	3.30	0.48	1.99	0.07	1.66	6.85	1.39	4.63
35-15-95	6.61	1,454.00	19,171.48	3,057.43	6,949.08	3.75	152.66	693.83	130.20	406.25	0.44	126.32	822.98	132.03	386.36
40-07-75	4.53	59.37	377.06	51.21	159.00	0.26	0.64	2.46	0.27	1.02	0.01	0.57	3.29	0.58	1.64
40-09-95	6.18	417.00	5,017.56	610.28	1,490.31	1.65	58.30	398.04	58.47	148.19	0.27	16.87	99.82	16.26	48.26
40-10-65	0.84	17.87	115.06	15.88	52.73	0.37	0.79	1.12	0.21	1.03	0.01	0.14	0.71	0.13	0.38
50-10-65	2.52	24.56	84.64	16.34	64.04	0.67	1.42	2.06	0.34	1.88	0.01	0.12	0.58	0.09	0.28
50-10-75	22.79	1,352.41	24,507.34	3,085.42	4,404.50	1.70	299.78	1,595.95	311.53	965.27	0.19	38.58	195.59	40.51	120.84
50-10-95	9.97	89.36	1,059.59	91.95	200.20	1.98	22.36	145.65	25.65	73.16	0.14	9.38	39.38	7.72	25.55
Average	43.55	26,053.36	123,470.17	35,989.85	104,843.81	1.03	50.69	271.05	49.04	151.98	0.13	99.75	410.85	88.61	295.66
Median	2.52	41.43	377.06	51.21	148.43	0.26	0.97	3.44	0.50	1.86	0.01	1.01	4.84	0.90	2.67

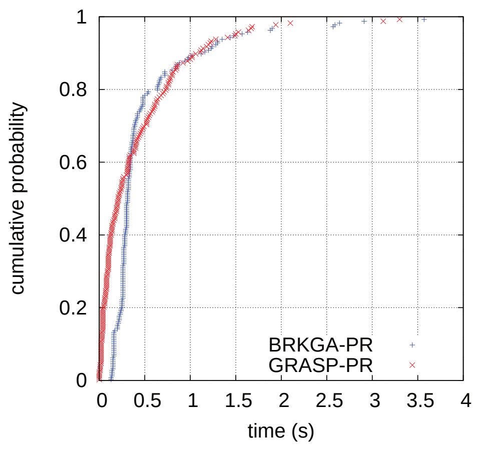
evidence that our proposed strategies have competitive performance since they are able to find the BKS in all runs in times that are significantly shorter than that those reported in Mateus et al. (2010). We also observe that our strategies are robust in the sense the algorithms were developed for a specific problem and they also perform well in a general application without any modification. Finally, statistical tests similar to the one described in previous subsections comparing our two approaches shows that GRASP-PR is slightly faster than BRKGA-PR, especially for instances with 35 or more facilities. For instances with up to 30 facilities, the behavior is the reverse.

Figure 7 shows the time-to-target plots depicting the run time distributions for BRKGA-PR and GRASP-PR with target solution defined as BKS. We choose the same instances used to show the time-to-target for GRASP-PR in Mateus et al. (2010). We observe that the behavior, in general, is similar for BRKGA-PR and GRASP-PR, but the performance of one algorithm can be better than the other depending on the instance.

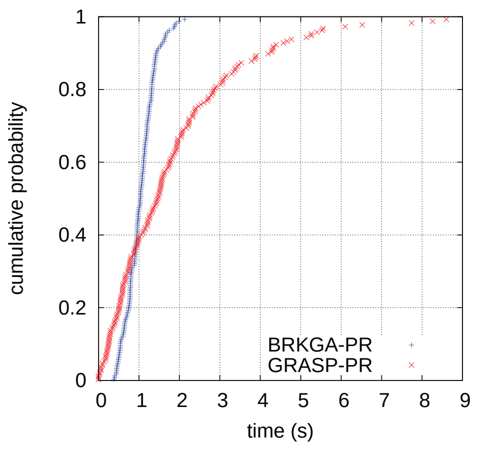
5. Concluding remarks

In this paper, we presented the problem of minimizing the cost of placing virtual machines across geo-separated data centers. A quadratic and two linear mathematical programming formulations were presented. Moreover, we presented several heuristics for solving this problem. In the experiments, we evaluate the performance of CPLEX using the proposed mathematical programming formulations, and the proposed heuristic methods.

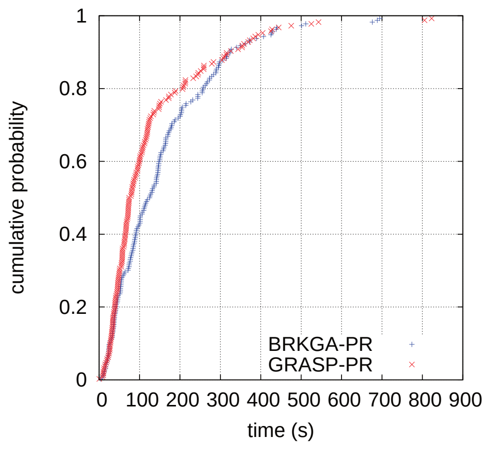
The results of CPLEX show that by adding the set of cuts the solver significantly improves the quality of the lower bounds for the LMVMPII model. We also observed that for this model, CPLEX can handle larger instances than considering LMVMP model, obtaining



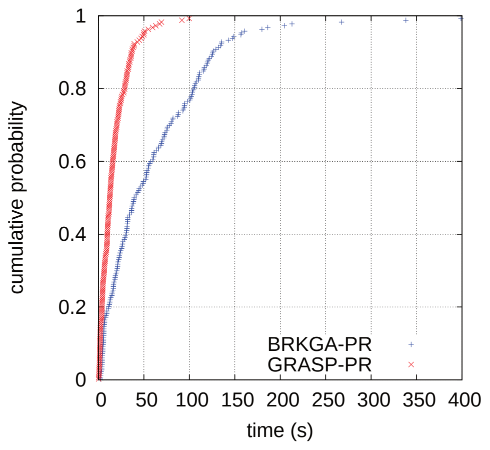
(a) TTT plot for `20-15-75`.



(b) TTT plot for `30-20-35`.



(c) TTT plot for `35-15-95`.



(d) TTT plot for `40-09-95`.

Figure 7: Cumulative probability distribution for BRKGA-PR and GRASP-PR running times for instances `20-15-75`, `30-20-35`, `35-15-95`, and `40-09-95`.

better lower bounds and feasible solutions in less computational time. However, as an exact method, CPLEX is limited to solve small instances, showing that heuristic methods are required on larger size instances.

Two heuristics approaches are used to solve the problem, GRASP and BRKGA. In both methods, we use a path-relinking procedure as an intensification mechanism. Also, we use a local search that explores many neighborhood strategies. Both heuristic approaches had similar performance, with the best performance achieved using local search `3V` and path-relinking `PRM`. A slight advantage for BRKGA was observed in small instances while GRASP has a slight advantage on larger instances. The same performance was observed when our strategies were applied to GQAP instances. In this case, our methods outperforms the previous state-of-the-art algorithm.

Finally, considering the high cost involved in this kind of problem, and the difficulty to obtain feasible solutions when taking into account several constraints and limited resources, the proposed algorithms are good alternative to reduce costs, while maintaining the reliability and the demand requirements of data centers.

6. Acknowledgments

This work has been partially supported by CAPES, CNPq project 462425/2014-2, PRH PB-217 Petrobras S.A. from Brazil. We thank to K. Joshi at AT&T Labs-Research for many helpful discussions. The work of Vaneet Aggarwal and Mauricio G.C. Resende was done when both were employed at AT&T Labs Research.

References

Aiex, R. M., Resende, M. G. C., and Ribeiro, C. C. (2007). TTT plots: A perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366.

Alicherry, M. and Lakshman, T. (2012). Network aware resource allocation in distributed clouds. In *2012 Proceedings IEEE INFOCOM*, pages 963–971. IEEE.

Armentano, V. A., Shiguemoto, A., and Løkketangen, A. (2011). Tabu search with path relinking for an integrated production–distribution problem. *Computers & Operations Research*, 38(8):1199–1209.

Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. (2011). Towards predictable datacenter networks. *ACM SIGCOMM Computer Communication Review*, 41(4):242.

Basseur, M., Seynhaeve, F., and Talbi, E.-G. (2005). Path Relinking in Pareto Multi-objective Genetic Algorithms. In Coello Coello, C., Hernández Aguirre, A., and Zitzler, E., editors, *Evolutionary Multi-Criterion Optimization SE - 9*, volume 3410, pages 120–134. Springer Berlin Heidelberg.

Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *INFORMS Journal on Computing*, 6(2):154–160.

Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D., and Silvera, E. (2012). A Stable Network-Aware VM Placement for Cloud Systems. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506. IEEE.

Cordeau, J. F., Gaudioso, M., Laporte, G., and Moccia, L. (2006). A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing*, 18(4):433–443.

Feo, T. A. and Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71.

Feo, T. A. and Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133.

Feo, T. A., Resende, M. G. C., and Smith, S. H. (1994). A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set.

Festa, P., Pardalos, P., Resende, M., and Ribeiro, C. (2002). Randomized heuristics for the Max-Cut problem. *Optimization Methods and Software*, 17(6):1033–1058.

Festa, P. and Resende, M. G. C. (2013). Hybridizations of GRASP with Path-Relinking. In Talbi, E.-G., editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 135–155. Springer Berlin Heidelberg.

Fischer, A., Botero, J. F., Beck, M. T., de Meer, H., and Hesselbach, X. (2013). Virtual Network Embedding: A Survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906.

Fischetti, M. and Monaci, M. (2014). Exploiting Erraticism in Search. *Operations Research*, 62(1):114–122.

Frieze, A. and Yadegar, J. (1983). On the quadratic assignment problem. *Discrete Applied Mathematics*, 5(1):89–98.

Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. (1997). Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. In Barr, R., Helgason, R., and Kennington, J., editors, *Interfaces in Computer Science and Operations Research*, volume 7 of *Operations Research/Computer Science Interfaces Series*, pages 1–75. Springer US.

Glover, F. (2014). Exterior Path Relinking for Zero-One Optimization. *International Journal of Applied Metaheuristic Computing*, 5(3):1–8.

Glover, F. and Laguna, M. (1993). Tabu Search. In Reeves, C. R., editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. Blackwell Scientific Publishing, Oxford, England.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, Vol. 29, n(3):653–684.

Glover, F., Laguna, M., and Martí, R. (2003). Scatter Search and Path Relinking: Advances and Applications. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 1–35. Springer US.

Gonçalves, J. F. and Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. a., Patel, P., and Sengupta, S. (2009). VL2: A Scalable and Flexible Data Center Network. *ACM SIGCOMM Computer Communication Review*, 39(4):51.

Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. (2010). SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *International Conference on - Co-NEXT, 10*, page 12, New York. ACM.

Guo, T., Shenoy, P., Ramakrishnan, K. K., and Gopalakrishnan, V. (2017). Latency-aware virtual desktops optimization in distributed clouds. *Multimedia Systems*, 0(0):22.

Hansen, P., Mladenović, N., and Moreno Pérez, J. A. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407.

Hoos, H. H. and Stützle, T. (1998a). Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In *Conference on Uncertainty in Artificial Intelligence, 14*, pages 238–245, San Francisco. Morgan Kaufmann Pub.

Hoos, H. H. and Stützle, T. (1998b). On the empirical evaluation of Las Vegas algorithms. Technical report, Computer Science Department, University of British Columbia.

Kaufman, L. and Broeckx, F. (1978). An algorithm for the quadratic assignment problem using Bender's decomposition. *European Journal of Operational Research*, 2(3):207–211.

Koopmans, T. C. and Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76.

Laguna, M. and Martí, R. (1999). GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11(1):44–52.

Lee, C. G. and Ma, Z. (2004). The generalized quadratic assignment problem. Technical report, Department of Mechanical and Industrial Engineering at the University of Toronto, Toronto, Ontario, M5S 3G8, Canada.

Martí, R., Laguna, M., and Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372.

Martí, R., Resende, M. G. C., and Ribeiro, C. C. (2013). Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226(1):1–8.

Mateus, G. R., Resende, M. G. C., and Silva, R. M. A. (2010). GRASP with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics*, 17(5):527–565.

Mittelmann, H. and Salvagnin, D. (2015). On solving a hard quadratic 3-dimensional assignment problem. *Mathematical Programming Computation*, pages 1–16.

Oliveira, C. A. S., Pardalos, P. M., and Resende, M. G. C. (2004). GRASP with Path-Relinking for the Quadratic Assignment Problem. In Ribeiro, C. C. and Martins, S. L., editors, *Experimental and Efficient Algorithms*, volume 3059 of *Lecture Notes in Computer Science*, pages 356–368. Springer Berlin Heidelberg.

Pessoa, A. A., Hahn, P. M., Guignard, M., and Zhu, Y.-R. (2010). Algorithms for the generalized quadratic assignment problem combining Lagrangean decomposition and the Reformulation-Linearization Technique. *European Journal of Operational Research*, 206(1):54–63.

Piao, J. T. and Yan, J. (2010). A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 87–92. IEEE.

Resende, M. G. and Ribeiro, C. C. (2016). *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer New York.

Resende, M. G. C. and Ribeiro, C. C. (2005). GRASP with Path-Relinking: Recent Advances and Applications. In *Metaheuristics: Progress as Real Problem Solvers*, volume 1, pages 29–63. Springer-Verlag, New York.

Resende, M. G. C. and Ribeiro, C. C. (2010). Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 283–319. Springer US.

Resende, M. G. C. and Ribeiro, C. C. (2014). GRASP: Greedy Randomized Adaptive Search Procedures. In Burke, E. K. and Kendall, G., editors, *Search Methodologies*, pages 287–312. Springer US.

Resende, M. G. C., Ribeiro, C. C., Glover, F., and Martí, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of Metaheuristics*, pages 87–107. Springer.

Spears, W. M. and DeJong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236.

Stefanello, F., Aggarwal, V., Buriol, L. S., Gonçalves, J. F., and Resende, M. G. C. (2015a). A Biased Random-key Genetic Algorithm for Placement of Virtual Machines across Geo-Separated Data Centers. In *Conference on Genetic and evolutionary computation, 15*, pages 1–8, Madrid. ACM.

Stefanello, F., Buriol, L. S., Aggarwal, V., and Resende, M. G. C. (2015b). A New Linear Model for Placement of Virtual Machines across Geo-Separated Data Centers. In *Simpósio Brasileiro de Pesquisa Operacional, 47*, pages 1–11, Porto de Galinhas, PE. Sociedade Brasileira de Pesquisa Operacional.

Toso, R. and Resende, M. (2015). A C++application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93.

Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 38(1-2):57–67.

Xie, D. and Hu, Y. C. (2012). The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers. In *Sigcomm, 12*, SIGCOMM '12, pages 199–210, New York. ACM.