A Biased Random-key Genetic Algorithm for Placement of Virtual Machines across Geo-Separated Data Centers

Fernando Stefanello
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
fstefanello@inf.ufrgs.br

Vaneet Aggarwal
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907 USA
vaneet@purdue.edu

Luciana S. Buriol
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
buriol@inf.ufrgs.br

José F. Gonçalves
Faculdade de Economia
Universidade do Porto
Porto, Portugal, 4200-464
jfgoncal@fep.up.pt

Mauricio G. C. Resende
Mathematical Optimization and Planning (MOP)
Amazon.com
Seattle, WA 98109 USA
resendem@amazon.com

ABSTRACT

Cloud computing has recently emerged as a new technology for hosting and supplying services over the Internet. This technology has brought many benefits, such as eliminating the need for maintaining expensive computing hardware and allowing business owners to start from small and increase resources only when there is a rise in service demand. 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, 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). We formalize the problem and propose a novel algorithm based on a biased random-key genetic algorithm (BRKGA) to find near-optimal solutions for the problem. The experimental results show that the proposed algorithm is effective in quickly finding feasible solutions and it produces better results than a baseline aproach provided by a commercial solver and a multi-start algorithm.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence] [**Problem Solving, Control Methods,and Search**]: [Heuristic methods]

Keywords

Combinatorial optimization; Cloud computing; Biased Random-Key Genetic Algorithm.

1. INTRODUCTION

Virtualization of physical servers have 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 depends, among other factors, on cost and performance. In order 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. Having all these performance guarantees for the application can help give an optimized service to the users. 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. This paper considers this placement problem, and gives novel solutions using a class of genetic algorithms.

Unfortunately, today's public cloud platforms such as Amazon EC2 [2] 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 today's data center networks (e.g., [6]), network bandwidth is a scarce resource shared across many tenants. In order to meet the reliability and the demand requirements, the data centers have to be placed all across 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 the major cost, which has not been accounted in the prior works on virtual machine placement to the best of our knowledge. With the limited bandwidth links between the data centers, networking intensive phases of applications collide and compete for the scarce network resources, which leads to their running times become unpredictable. 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.

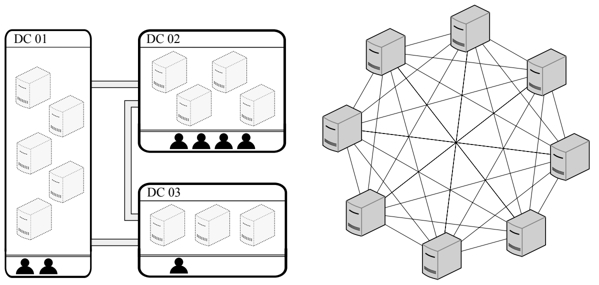
Placement of virtual machines within a data center have been widely explored [7, 3, 16]. These papers account for the networking needs in addition to the CPU and memory needs within a data center. For example, [7] proposes bandwidth reservation between every pair of VMs. [3] proposes a simpler virtual cluster (VC) model where all virtual machines are connected to a virtual switch with links of bandwidth B. [16] extends these approaches to consider time-varying network requirement. However, all these works account for a single data center where the bandwidths are much larger as compared to the bandwidths across data centers. Instead, this paper deals with the placement of virtual machines across geo-separated data centers

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. In order to meet the application's quality of service guarantees, there is a required 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 for the application.

This problem is a generalization of the NP-hard Generalized Quadratic Placement Problem given in [11]. Solving this problem optimally is possible only in very small instances, which may not represent the size found in real-world applications. Thus, we propose a Biased Random-key Genetic Algorithm (BRKGA) for solving the Virtual Machine Placement Problem. We test the performance of the proposed algorithm in a dataset comprised of instances with sizes ranging from small to large. We show that the algorithm is able to quickly find feasible solutions, producing better results than a classic multi-start algorithm and an exact approach using CPLEX [10].

The paper brings several contributions: i) We model a multi-data center virtual machine placement problem with quality of service guarantees for the application; ii) The quadratic problem with integer constraints is reduced to a mixed integer linear program for placement of virtual machines; iii) Novel algorithms based on BRKGA are proposed to place virtual machines across multiple data centers; iv) A generator of instances which contain at least one feasible solution is provided; v) We provide experimental results for CPLEX and the BRKGA applied on the proposed instances.

The rest of the paper is organized as follows. In Section 2, we present mathematical models for the Virtual Machine Placement Problem in multiple data centers. The biased random-key genetic algorithm with local search is presented in Section 3. Computational results are presented in Section 4. Finally, conclusions are drawn in Section 5.



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

Figure 1: Input data representation.

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.

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 are 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 required latency t_{uv} for each VM v.

Figure 1 shows a representation of the input data components: the data centers (Figure 1a), and the virtual machines (Figure 1b). The first component is composed by 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 by eight virtual machines, where each link represents the bandwidth and required latency.

The performance of mixed integer linear programming solvers has improved considerably over the last few years. IBM ILOG CPLEX Optimizer [10] is a general-purposed black-box solver based on the state-of-the-art exact algorithms for integer programming and has been successfully applied in many combinatorial optimization problems. In order to investigate the CPLEX performance and provide baseline results for comparison of heuristic methods, we present a quadratic and a linear mathematical model for the VMPlacement problem. Results are provided in Section 4.

Next we present the quadratic mathematical model for the VMPlacement (QMMVMP).

Parameters:

N :	set of data centers;
K :	set of virtual machines;
U :	set of users;
a_i :	capacity in number of VMs DC i can host;
B_{ij} :	bandwidth between DCs i and j;
L_{ij} :	latency between DCs i and j;
Cij :	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 which hosts user u;
t_{vu} :	required latency between user u and VM v.

The binary decision variable x_{iv} is set to one when VM v is located into DC i, and zero otherwise.

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

subject to:

$$\sum_{v\in K} x_{iv} \le a_i \qquad \forall\ i \in N, \tag{1b}$$

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

$$\sum_{v\in K}\sum_{w\in K} x_{iv} x_{jw} b_{vw} \le B_{ij} \qquad \forall\ i,j \in N, \tag{1d}$$

$$\sum_{i\in N}\sum_{j\in N} x_{iv} x_{jw} L_{ij} \le l_{vw} \qquad \forall\ v,w \in K, \tag{1e}$$

$$\sum_{i\in N} x_{iv} L_{i,d(u)} \le t_{vu} \qquad \forall\ u \in U,\ \forall\ v \in K, \tag{1f}$$

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

Objective function (1a) minimizes the cost of placing each pair of virtual machines v and w to DCs i and j. Constraints (1b) require that the number of VMs in each DC must not exceed the DC capacity. Constraints (1c) require that each VM must be assigned to exactly one DC. Constraints (1d) 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 (1e) require that the latency required between each pair of VMs should be respected, i.e, if VMs v and w are placed respectively to DCs i and j, then the latency between DCs i and j should not exceed the required latency between VMs v and w. Constraints (1f) 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 on a 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 (1g) define the variables domain.

The VMPlacement is an generalization of the NP-hard Generalized Quadratic Assignment Problem (GQAP), formulated in [11]. Thus, based on model $L3$ from [11], we present a mixed-integer 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 mixed-integer linear mathematical model for VMPlacement named as LMMVMP, can be formulated as the following:

$$\min \sum_{i\in N}\sum_{j\in N}\sum_{v\in K}\sum_{w\in K} C_{ij} b_{vw} y_{ivjw} \tag{2a}$$

subject to:

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

$$\sum_{i\in N} x_{iv} = 1 \qquad \forall\ v \in K, \tag{2c}$$

$$\sum_{i\in N} y_{ivjw} = x_{jw} \qquad \forall\ v,w \in K,\ \forall\ j \in N, \tag{2d}$$

$$y_{ivjw} = y_{jwiv} \qquad \forall\ v,w \in K,\ \forall\ i,j \in N, \tag{2e}$$

$$\sum_{v\in K}\sum_{w\in K} y_{ivjw} b_{vw} \le B_{ij} \qquad \forall\ i,j \in N, \tag{2f}$$

$$\sum_{i\in N}\sum_{j\in N} y_{ivjw} L_{ij} \le l_{vw} \qquad \forall\ v,w \in K, \tag{2g}$$

$$\sum_{i\in N} x_{iv} L_{i,d(u)} \le t_{vu} \qquad \forall\ u \in U,\ \forall\ v \in K, \tag{2h}$$

$$x_{iv} \in \{0,1\} \qquad \forall\ i \in N,\ \forall\ v \in K, \tag{2i}$$

$$0 \le y_{ivjw} \le 1 \qquad \forall\ i,j \in N,\ \forall\ v,w \in K. \tag{2j}$$

The LMMVMP is obtained by replacing the product $x_{iv}x_{jw}$ by y_{ivjw} from QMMVMP. In addition four sets of constraints are added. Constraints (2d) and (2e) define the relation between variables x and y. Constraints (2e) also impose the symmetry relation to variables y. Finally, constraints (2j) define the domain of variables y.

We note that the model QMMVMP has quadratic constraints, while LMMVMP not. The objective function also changes from a quadratic function in QMMVMP to a linear function in LMMVMP. However, the mixed-integer linear problem LMMVMP has a considerable higher number of variables, having variables y_{ivjw} in addition to the previous variables x_{iv}. Thus, the number of variables change from $O(NK)$ in QMMVMP to $O(N^2K^2)$ in LMMVMP. We note that if the optimal solution of LMMVMP is (x^*_{iv}, y^*_{ivjw}), then (x^*_{iv}) is the optimal solution for QMMVP. The proof that both models are equivalent can be easily obtained by extending the proof for QAP provided in [11].

3. A BIASED RANDOM-KEY GENETIC ALGORITHM

A biased random-key genetic algorithm (BRKGA) is a metaheuristic for finding optimal or near-optimal solutions for hard combinatorial optimization problems [5]. A BRKGA is a class of genetic algorithms based on random keys, where solutions are encoded as a 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 *decoders*. 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*).

Figure 2 [5] shows a flowchart of a general scheme of a BRKGA, that 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

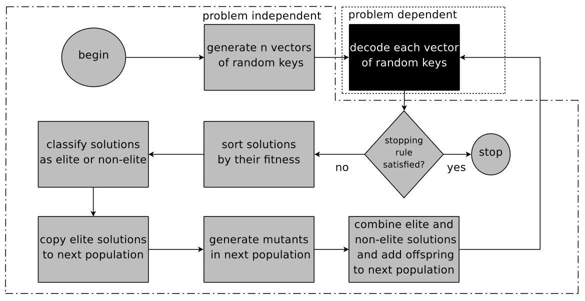

problem independent
problem dependent
begin
generate n vectors of random keys
decode each vector of random keys
classify solutions as elite or non-elite
sort solutions by their fitness
stopping rule satisfied?
no
yes
stop
copy elite solutions to next population
generate mutants in next population
combine elite and non-elite solutions and add offspring to next population

Figure 2: Flowchart of a BRKGA.

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 fittest p_e solutions in the population, and another larger set of $p - p_e > p_e$ non-elite solutions. Population $g+1$ is generated as follows. All p_e elite solutions of population g are copied without changing to population $g+1$. This elitist strategy maintains the best solutions on hand. To ensure that mutation is present in the evolution, p_m *mutants* are added to population $g+1$. A mutant is simply a vector of random keys, generated in the same fashion than an initial solution.

With $p_e + p_m$ solutions accounted for population $g+1$, $p - p_e - p_m$ additional solutions must be generated to complete the p solutions that make up population $g+1$. This is done through *mating* or *crossover*. A parent-A is selected randomly from the elite solutions, and the parent-B parent is selected randomly between the set of non-elite solutions. A child C is produced by combining the parents using parameterized uniform crossover. 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 $i = 1, \ldots, n$.

Figure 2 from [5] shows the *problem-independent* and *problem-dependent* components of a BRKGA. The independent components are applied without any acknowledgement 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 were made, and how the algorithm stops. We next describe the encoding and decoding procedures for the proposed algorithm, and give values for parameters and stopping criterion in Section 4.

3.1 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 to the DC which produces the least increase in the objective function.

The decoder starts with a list whose 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 index of virtual machines is used as an order in which virtual machines should be placed. Following this order, the next step is to place each virtual machine v to a DC. This is done by placing virtual machine v in DC i which produces the least increase in the objective function. Note that the cost to insert the VM v in each DC considers the previous virtual machines placed. When all VMs are placed, the decoder returns the fitness value for the respective 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 to DC $\lfloor k_i * N \rfloor$.

Place a virtual machine to a data center can violate some of the constraints. In order to alleviate this problem, 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 added by a sufficiently large number M for each violated constraint. This penalization strategy is applied whenever a solution is evaluated, including in both decoders, and in both local search strategy describe in the next subsection. In our experiments we use $M = 10^{10}$.

3.2 Local search

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 try to replace the current solution by a better neighbor solution, until no improvement can be reached. A neighbor solution can be obtained by applying moves defined by a neighborhood structure. Two classical neighborhood structures are used to obtain neighbor solution, namely *shift* and *swap*.

A shift operation moves a virtual machine from the current data center to a different data center. In a *shift search*, we test all shift moves selecting virtual machines in a circular order of their indexes (starting from index zero), and calculating the cost to remove and insert the virtual machine in all others data centers, also chosen in a circular order of index. Once an improvement is reached, the virtual machine is shifted to the new data center and the search continues considering the next data center. The procedure stops when no shift move can improve the solution.

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 i and j in a circular order of their indexes (starting from index zero). When an improvement is reached, the virtual machines positions are interchanged and the search continues by selecting a next virtual machine. Moves where $j \leq i$ and i and j are in the same data center are not evaluated. The procedure ends when no swap move can improve the solution.

The local search is applied after each decoder. In our experiments, we evaluated the performance of two local search strategies. The first, called *LSS*, considers only the shift search. The second, called *LSW*, considers shift search and swap search applied sequentially, until no improvement is reached in both searches. Note that using a penalization strategy described in the previous subsection, the local search is also applied to infeasible solutions.

4. COMPUTATIONAL RESULTS

The experiments were conducted on a cluster with quad-core Intel Xeon E5530 2.4 GHz CPUs, with at least 48 GB of RAM running GNU/Linux. BRKGA was implemented in C++, using the API described in [15] and a commercial solver IBM ILOG CPLEX Optimizer version 12.6.0.0 (C++ API) was used to evaluate the mathematical model. 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 CPLEX performance in order to obtain a lower bound, analyze which size of instance the solver can handle, and obtain baseline results for comparison of heuristic methods. The second was to evaluate the performance of the proposed BRKGA, comparing two decoders, the impact of the local improvement procedures when embedded in the decoders, and comparing it with a simple multi-start algorithm [13]. We next describe the method to generate the dataset used in the experiments.

4.1 Data set

In this subsection we present an instance generator that we proposed and implemented to generate the data set used to evaluate CPLEX and BRKGA for VMPlacement. For generating each instance the generator receives as input four parameters: $|N|$, $|K|$, $|U|$, and P (the latter 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 a capacity of each data center, each set of pre-placed $s \in \mathcal{S}$ is generated by randomly placing each virtual machine to 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.

Let a random numbers generator by uniform distribution, and M' be 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 machine $v \in K$ 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 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 instances sizes, while parameters $|U|$ and P can be used to adjust how the problem should be restricted. Finally, we generate 36 instances by combining values from $N = \{10, 25\}$, $K = \{25, 50, 100, 150, 200\}$, $U = \{K_i * 0.5, K_i, K_i * 1.5\}$, and $P = \{70, 90\}$. Table 2 contains 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/`.

4.2 CPLEX results

In the first experiment we evaluate the performance of CPLEX with the mathematical models described in Section 2. We used the standard CPLEX solvers for models QMMVMP and LMMVMP. The running time limit was set to one day (86,400 seconds) and the number of threads was set to one. The remaining parameters were maintained on the default values. The CPLEX performance with LMMVMP was considerably better than with QMMVMP model, and for this reason we report only results for the mixed-integer linear model.

Table 1 shows CPLEX results. The first column shows the name of instances. The second column (BKS) shows the objective function of best known solution value for each instance. The next three columns show respectively the values of lower bound, best integer solution and the percentage gap returned by CPLEX. Finally the last column shows the percentage gap between the best integer solution and BKS.

We can draw three main observations from this experiment. First, we omitted the results for instances with 25 data centers because CPLEX spent the whole time in the presolve phase without solving the root relaxation node. This shows that CPLEX cannot be applied with this mathematical model to large instances. Second, CPLEX gaps are still high after 24h of computation. We think this is due to the low relaxation quality of this model. Finally, for the set of instances with 10 data centers, CPLEX was able to find at least one feasible solution (column Integer Solution), in many cases, even before starting a node exploration of the branch-and-bound. However, the solver still has a high gap from BKS.

Table 1: CPLEX detailed results.

Instance	BKS	Lower Bound	Integer Solution	CPLEX GAP	BKS GAP
10_025_012_70	114,582.50	67,006.63	116,264.44	42.37	1.47
10_025_012_90	84,461.30	40,592.25	88,087.12	53.92	4.29
10_025_025_70	90,997.90	65,997.32	93,729.48	29.59	3.00
10_025_025_90	124,763.66	86,281.96	125,365.26	31.18	0.48
10_025_037_70	100,801.80	79,139.70	104,350.38	24.16	3.52
10_025_037_90	106,617.94	81,678.77	107,558.00	24.06	0.88
10_050_025_70	414,689.30	74,248.43	442,548.40	83.22	6.72
10_050_025_90	460,414.96	117,147.62	480,146.00	75.60	4.29
10_050_050_70	360,102.12	116,523.45	374,071.02	68.85	3.88
10_050_050_90	403,272.24	113,953.64	420,173.88	72.88	4.19
10_050_075_70	349,135.78	166,802.77	362,853.92	54.03	3.93
10_050_075_90	500,668.88	233,743.38	513,161.02	54.45	2.50
10_100_050_70	1,677,015.90	159,285.07	1,884,262.62	91.55	12.36
10_100_050_90	1,804,385.34	200,022.30	1,916,126.76	89.56	6.19
10_100_100_70	1,465,034.60	213,163.82	1,546,897.78	86.22	5.59
10_100_100_90	2,145,917.62	389,652.74	2,256,408.46	82.73	5.15
10_100_150_70	1,572,976.60	367,091.18	1,702,573.74	78.44	8.24
10_100_150_90	1,858,242.74	575,983.27	1,968,341.96	70.74	5.92
Average				61.86	4.59

In summary, CPLEX was not able to prove the optimality of any instance and presented on average a high percentage gap of 4.59 %. Thus, these results motivated us to propose heuristic solutions to solve the problem.

4.3 Multi-start and feasibility results

This section presents results and analysis of a greedy constructive heuristic combined with a local search in a multi-start algorithm (MS). The main goals of the following experiments are two-fold: to show the difficulty of finding feasible solutions using a constructive heuristic, and to explore the effect of embedding a local search procedure in order to obtain a feasible solution.

In order to support our analysis, we evaluate the performance of a greedy constructive heuristic embedded in a multi-start algorithm. The greedy constructive heuristic starts from a random order of virtual machines and uses the same idea of decoder D1 to generate greedy solutions, i.e, VMs are placed in the DC that produces the least increase in the objective function. The multi-start algorithm repeats the constructive heuristic until the stop criteria is reached. Once a solution is obtained by the constructive algorithm, the local search is applied until reaching a local minimum.

We run three algorithms on each instance. The stopping criteria of each run was a time limit of $|K| * |N| * \theta$ seconds, where $\theta = 0.8$. The algorithms are the multi-start algorithm without applying local search (NoLS), by applying the local search with only shift moves (LSS), and by applying both moves (LSW). The values in Table 2 refer to the percentage of solutions evaluated that were feasible. Results represent an average over 10 runs.

The first observation is that the probability of finding a feasible solution using only a greedy constructive heuristic without local search is low. In most instances, the percentage of feasible solutions was less than 0.2%, and the average over all instances was slightly more than 1%. In five cases, no feasible solution was found, as for example in instance `25_200_300_90`, even haven evaluated 529,769 solutions.

With LSS the percentage of feasible solutions increased considerable in comparison with NoLS. The number of evaluated solutions decreased around 42%, but the percentage of feasible solutions increased to more than 21%. When applying the local search LSW, the percentage of feasible solutions increased even more, and on average, more than three in four evaluated solutions are feasible. However, the number of solution evaluated decreased to around 2% of the number of solutions evaluated by LSS. This results show that a large neighborhood helps to improve the feasibility, even if the local search procedure is applied few times.

Table 2: Percentage of feasible solutions found by the multi-start algorithm with the greedy constructive heuristic.

Instance	NoLS	LSS	LSW	Instance	NoLS	LSS	LSW
10_025_012_70	7.95	54.26	91.99	25_100_050_70	3.70	58.54	95.41
10_025_012_90	1.49	22.43	93.03	25_100_050_90	0.00	8.49	79.85
10_025_025_70	14.88	84.55	99.65	25_100_100_70	0.24	25.29	83.61
10_025_025_90	0.03	4.13	51.59	25_100_100_90	0.00	1.37	61.83
10_025_037_70	3.23	57.01	78.97	25_100_150_70	0.41	27.45	90.82
10_025_037_90	0.02	3.64	47.14	25_100_150_90	0.00	0.98	60.82
10_050_025_70	1.48	33.85	95.35	25_150_075_70	0.90	40.04	91.46
10_050_025_90	0.05	15.32	90.57	25_150_075_90	0.00	3.25	84.98
10_050_050_70	0.03	15.62	63.60	25_150_150_70	0.08	17.25	78.37
10_050_050_90	0.02	10.55	60.16	25_150_150_90	0.00	2.10	58.95
10_050_075_70	0.41	53.26	77.68	25_150_225_70	0.12	20.19	80.31
10_050_075_90	0.01	6.15	66.07	25_150_225_90	0.00	0.67	47.80
10_100_050_70	0.17	40.90	98.71	25_200_100_70	0.01	11.69	80.63
10_100_050_90	0.00	4.09	82.51	25_200_100_90	0.00	15.82	95.59
10_100_100_70	1.40	58.51	90.48	25_200_200_70	0.02	12.90	76.35
10_100_100_90	0.00	5.46	87.55	25_200_200_90	0.00	8.73	92.86
10_100_150_70	0.07	34.88	83.72	25_200_300_70	0.00	7.81	61.87
10_100_150_90	0.00	0.43	64.05	25_200_300_90	0.00	1.72	85.64
Average	1.74	28.06	79.05		0.30	14.68	78.18

Even with a large 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 instance generator produces instances not trivial to solve.

4.4 BRKGA results

This subsection presents results for the biased random-key genetic algorithm. First we describe experiments for tuning the parameters, and finally we present the main results obtained by the proposed BRKGA. We also compare results with the multi-start algorithm described in the previous subsection.

We used the *iterated racing* procedure [4] to tune the BRKGA parameters. This method consists of sampling configurations from a particular distribution, evaluating them using either the Friedman test or the t-test, and refining the sampling distribution with repeated applications of F-Race. We uses the irace package [12], implemented in R [14].

For each heuristic, we used a budget of 2,000 experiments in the tuning procedure, and two digits for real number parameters. Twelve instances were selected from the original set in order to include the most different values. The stopping criteria for all runs was a time set to $|K| * |N| * \theta$ seconds (where $\theta = 0.8$). The following ranges are used in the tuning: p - population size $\in \{25, 50, 75\}$; p_e - elite percentage $\in [0.10, 0.30]$; p_m - percentage of mutants introduced at each generation $\in [0.05, 0.30]$; and ρ_A - probability of inheriting each key from elite parent $\in [0.5, 0.8]$. Finally, a restart parameter $r \in [50, 800]$ was used to restart the population after r generations without improvement in the incumbent solution.

The tuning was applied to four algorithm configurations, that include both decoders and both local search strategies. In all results returned by irace, values p_e, p_m, and ρ_A were similar and we adopted the elite size $p_e = 0.24p$, the set of mutants $p_m = 0.12p$, and the probability of inheriting $\rho_A = 0.6$. For the restart parameter, we decided to disable it when the local search LSW is used. This choice is supported by the fact that most of the processing time is spent in the local search procedure and the total number of generations tends to be small. For the case of LSS, the restart parameter was set to $r = 350$ for D1, and $r = 650$ for D2. Finally, for the population size parameter, irace suggested four sets of parameter setting, and the three populations

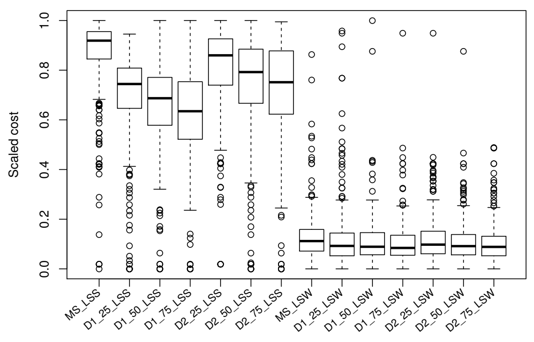


Figure 3: Dispersion of scaled cost for each algorithm.

sizes appear in the suggested values. For this reason, we decided to run a large experiment to analyse the best population size, comparing both local searches procedures and both decoders, and also to provide a comparison between the results of BRKGA and the results of MS obtained in the previous subsection. The experiment consists in evaluating the BRKGA in all instances running all combinations of decoders, local searches and population size. We made ten independent runs for each instance with the time limit set to $|K| * |N| * \theta$ seconds, where $\theta = 0.8$.

To compare the results with respect the cost, we first scale the results 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 analysed algorithms is scaled to 1 and the lowest is scaled to 0. 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 algorithm. The dots are the outliers. Algorithms are represented on the horizontal axis. BRKGA algorithms are encoded by the decoder, population size, and local search. Multi-start algorithms are encoded by MS plus type of local search.

Figure 3 shows that the results with local search LSW clearly overcome the results obtained with LSS. The results with $p = 75$ and decoder D1 tend to be better than the other approaches. Regarding the comparison of algorithms, BRKGA produces better results than MS.

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 3 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" overcome the "column algorithm", or the opposite in the case "greater".

Table 3 shows tests only with local search LSW since all test between LSS and LSW indicate that both strategies are statistically significant. This corroborate the analysis from Figure 3 showing that use LSW produce better results than use LSS.

Table 3: Values of medians, p-values, and difference in median location for cost distributions using a confidence interval of 99%.

	MS	D1_25	D1_50	D1_75	D2_25	D2_50	D2_75
MS	**0.112**	0.013	0.016	0.023	0.008	0.016	0.021
D1_25	<	**0.092**	0.003	0.006	-0.003	0.005	0.009
D1_50	<	0.427	**0.089**	0.006	-0.007	0.002	0.006
D1_75	<	0.087	0.065	**0.084**	-0.013	-0.003	0.001
D2_25	0.025	0.486	0.036	>	**0.098**	0.010	0.012
D2_50	<	0.181	0.484	0.366	<	**0.091**	0.003
D2_75	<	0.034	0.120	0.727	<	0.424	**0.088**

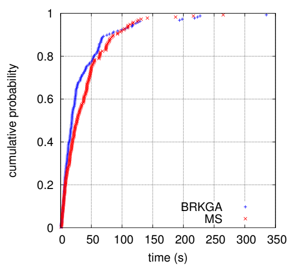
Table 3 shows that the difference between BRKGA and MS is statistically significant and BRKGA produces better results than MS, except for D2_25 where the test is inconclusive for a confidence interval of 99%. Tests between BRKGA algorithms indicate that the differences are not statistically significant, except in some cases where D2_25 performed worst. However, the medians and the difference in median location indicate that the best results are obtained with D1_75.

A set of statistical tests was also performed with results for instances with 10 data centers, considering the BRKGA with the default stopping criteria (maximum 800 seconds for the larger instance) and CPLEX running per 24h (Subsection 4.2). The tests confirm that any BRKGA approach produces better results than CPLEX.

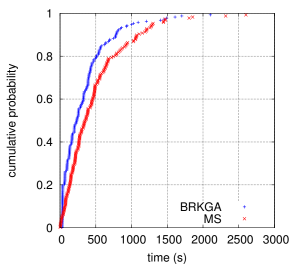
In summary, BRKGA with the parameters $p = 75$, $p_e = 0.24p$, $p_m = 0.12p$, $\rho_A = 0.6$, using local search LSW, and decoder D1 produced the better results for the instances evaluated in comparison with MS and CPLEX.

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 used by [1] and have been advocated by Hoos and Stützle [9, 8] as a way to characterize the running times of stochastic algorithms for combinatorial optimization. The experiment consists in performing 200 runs of BRKGA and MS algorithm for two instances until a timelimit is reached. The instances chosen was one at random from each group of data centers, and the target value was set as the higher result from all runs and both algorithms. Defined the target, we extract on each run the time to rearch a solution at least as good as the target, i.e 467,305 for `010_050_025_090` and 4,426,014 for `025_150_225_070`.

Figures 4a and 4b illustrate the cumulative probability plot obtained by using BRKGA and MS for two instances. For the instance `010_050_025_090`, we observe that the probability that the BRKGA finds a solution that is at least as good as the target value in less than 20 seconds is 50%, in less than 60 seconds is 80%, and in less than 120 seconds in 95%. For the instance `025_150_225_070`, the probability that the BRKGA finds a solution that is at least as good as the target value in less than 230 seconds in 50%, in less



(a) TTT plot for the instance 010_050_025_090.



(b) TTT plot for the instance 025_150_225_070.

Figure 4: Cumulative probability distribution.

than 500 second in 80%, and in less than 1020 seconds is 95%. In both instances, BRKGA overcomes MS obtaining higher probability in the same running time.

A final observation about running times of BRKGA is that results are reported using single-thread in order to provide a fair comparison with the MS and CPLEX results. However, the BRKGA API provides an efficient multi-thread decoding [15], that could be used to reduce substantially the running time when multiple processors are available.

5. CONCLUDING REMARKS

In this paper we presented the problem of minimizing the cost of virtual machines placement across geo-separated data centers. A quadratic and a linear mathematical formulation were presented. Moreover, a biased random-key genetic algorithm was proposed in order to find near-optimal solutions for the problem. Computational tests were conducted in a set of synthetic instances to evaluate the performance of CPLEX using the proposed mathematical formulations. Furthermore, the BRKGA was applied on the same set of instances. A set of experiments shows that BRKGA outperforms CPLEX significantly in terms of running times, achieving significantly better solutions in less time, and also produces better results than a simple multi-start algorithm. Thus, the proposed algorithm is a competitive algorithm which reduces costs while maintaining the reliability and the demand requirements of data centers.

6. ACKNOWLEDGMENTS

The authors would like to thank Kaustubh Joshi, Moo-Ryong Ra, and Alexander Varshavsky from AT&T Labs-Research, Bedminster NJ for helpful discussions.
This work has been partially supported by CAPES, CNPq project 462425/2014-2, PRH PB-217 Petrobras S.A. from Brazil; AT&T Labs Research from USA; and projects PTDC/EGE-GES/117692/2010 and NORTE-07-0124-FEDER-000057 funded by NPROP (ON.2 - O Novo Norte), under NSRF Framework, through ERDF and the Programme COMPETE, and by national funds through FCT from Portugal. The research of Mauricio G. C. Resende and Vaneet Aggarwal was done while they were employed at AT&T Labs Research.

7. REFERENCES

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

[2] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/. Last accessed: February, 2015.

[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253, New York, NY, USA, 2011. ACM Press.

[4] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated F-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer Berlin Heidelberg, 2010.

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

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

[7] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *Proceedings of the 6th International COnference on - Co-NEXT '10*, Co-NEXT '10, page 1, New York, NY, USA, 2010. ACM.

[8] H. H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 238–245, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[9] H. H. Hoos and T. Stutzle. On the empirical evaluation of Las Vegas algorithms. Technical report, CS Department, University of British Columbia, 1998.

[10] IBM ILOG CPLEX Optimizer. www.cplex.com. Last accessed: February, 2015.

[11] C. G. Lee and Z. Ma. The generalized quadratic assignment problem. Technical report, Department of Mechanical and Industrial Engineering at the University of Toronto, Toronto, Canada, 2004.

[12] M. Loopez-Ibanez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace Package: Iterated Race for Automatic Algorithm Configuration. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

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

[14] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013.

[15] R. Toso and M. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):1–15, 2014.

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