Received: 28 February 2016 | Revised: 11 August 2016 | Accepted: 12 October 2016

DOI 10.1002/nem.1958

WILEY

RESEARCH ARTICLE

A tree-based algorithm for virtual infrastructure allocation with joint virtual machine and network requirements

Ramon de Oliveira | Guilherme Piegas Koslovski

Graduate Program in Applied Computing, Santa Catarina State University, Joinville, Santa Catarina, Brazil

Correspondence
Guilherme Piegas Koslovski, Graduate Program in Applied Computing, Santa Catarina State University, Joinville, Santa Catarina, Brazil.
Email: guilherme.koslovski@udesc.br

Summary

Cloud providers have introduced the on-demand provisioning of virtual infrastructures (VIs) to deliver virtual networks of computing resources as a service. By combining network and computing virtualization, providers allow traffic isolation between hosted VIs. Taking advantage of this opportunity, tenants have deployed private VIs with application-optimized network topologies to increase quality of experience of final users. One of the main open challenges in this scenario is the allocation of physical resources to host VIs in accordance with quality of service computing (eg, virtual CPUs and memory) and network requirements (guaranteed bandwidth and specific network topology). Moreover, a VI can be allocated anywhere atop a network datacenter, and because of its NP-hard complexity, the search for optimal solutions has a limited applicability in cloud providers as requesting users seek an immediate response. The present work proposes an algorithm to accomplish the VI allocation by applying tree-based heuristics to reduce the search space, performing a joint allocation of computing and network resources. So as to accomplish this goal, the mechanism includes a strategy to convert physical and virtual graphs to trees, which later are pruned by a grouped accounting algorithm. These innovations reduce the number of comparisons required to allocate a VI. Experimental results indicate that the proposed algorithm finds an allocation on feasible time for different cloud scenarios and VI topologies, while maintaining a high acceptance rate and a moderate physical infrastructure fragmentation.

KEYWORDS

allocation, cloud computing, cloud networking, IaaS, NaaS, virtual infrastructures

1 | INTRODUCTION

Cloud computing comes at various shapes and flavors. Providers have delivered different layers of information technology (IT) infrastructures as dynamically provisioned services following a pay-as-you-go model. Specifically, Infrastructure as a Service (IaaS) and Network as a Service providers offer virtual machines (VMs), routers, and virtual networks as services, composing dynamically provisioned virtual infrastructures (VIs). This combination of cloud computing and cloud networking allows the creation of entirely virtualized infrastructures atop a network datacenter.[1] Basically, a VI is a fully virtualized infrastructure that appears to be physical but in reality shares the underlying substrate with other VIs during a given time frame.[2] By renting only the resources that they need, when they need, users can reduce their costs with IT infrastructure as they no longer need to spend on buying and managing their own servers. In fact, more and more companies have opted to outsource their IT infrastructure to cloud providers.[3]

Cloud applications performance can critically depend on the underlying network, and without mechanisms to strict network reservation, the execution is subject to high variations. Even in short periods, the bandwidth available to an application can differ by a factor of 5 or more.[4] In the same vein, Persico et al[5] highlighted that network allocation strategies can heavily impact the performance of applications hosted on public cloud providers. Although some VM instance types have a fixed CPU performance and dedicated

devices, the virtual network interconnection is still a determinant for cloud applications performance.

With VIs, the requesting user is granted full control over the aggregation of IT and network resources and is kept aside from management and maintenance of physical resources.[6,7] Several projects and companies have studied mechanisms to create, allocate, deploy, and manage VIs, exploring technologies such as virtualization of machines, links, switches, and routers for composing and providing isolated VIs. Indeed, for delivering VIs, providers must allocate* physical resources to host the virtual objects that constitute the VI.[8,9] Formally, VIs and network datacenter can be seen as graphs: The vertices denote routers, switches, VMs, and servers while the edges represent the communication links. From this angle, the provider must find a mapping between virtual and physical graphs. This task belongs to the set of NP-hard problems as other problems already proven to be in this set may be reduced to it.[10]

Specialized literature comprises approaches to find embedding solutions.[11–15] Although effective in their fields, they mostly solve the allocation of virtual networks (Virtual Network Embedding [VNE]), ignoring the existence of VMs. Even considering the allocation of physical resources for hosting VIs as a graph formulation, there is a specialization of datacenter vertices functionality that must be addressed. A physical candidate for hosting a virtual switch or router must have forwarding functionality (a traditional VNE scenario) while candidates for hosting VMs have specific attributes (e.g., CPU and memory). Such functionalities are usually represented as vertices and edges attributes specified by tenants.[16] As traditional VNE approaches focus on allocation of virtual networks atop networking equipment (router and switches), the number of hosting candidates is usually reduced when faced to a cloud datacenter composed of many servers.

Virtual Network Embedding–based formulations consist in finding paths between source-destination pairs respecting the bandwidth requirements. Although some approaches perform a joint allocation of vertices and edges, commonly, the location of source and destination nodes is known in advance by the mechanism. Indeed, some proposals have focused on the allocation of VMs only, without considering the communication among them.[17–19] Moreover, the search for optimal solutions has limited applicability as cloud tenants need an immediate response from providers.

Most proposals do not necessarily consider physical and virtual topologies that represent cloud providers scenarios (e.g., applications with multitiered architecture, virtual private clouds (VPCs), hierarchical network topologies, and datacenter topology), a limitation that may influence the acceptance rate of VI requests.[20,21] Analysis and simulation with random network requests (only considering the probability of nodes interconnection) limit the scope and decrease the applicability of such proposals in cloud datacenters. On cloud datacenters, a network topology usually has a subset of redundant links [22,23] while virtual requests have topologies guided by hosted application requirements.[24]

Focusing on VI allocation atop cloud datacenters with joint VMs and network requirements, we propose a heuristic having as premise that both graphs (VI and physical) are represented in the form of trees (the graphs are connected and acyclic). Following this premise, strategies to reduce search space by grouping subtrees information are applied, which consequently accelerates the allocation process. The mechanism receives a VI request of any virtual topology and converts it to a tree. Although intuitive and well-studied in graph theory fields, the conversion of VIs in trees is not a trivial procedure as the algorithm is not allowed to change the requested communication pattern. For example, virtual links requesting quality-of-service parameters must be correctly allocated and delivered by the provider to respect the service level agreement (SLA) established with tenants. Succinctly, a conversion is allowed only if it complies with the SLA. The conversion of a network datacenter topology in a tree follows a different approach. As network datacenters are usually over-provisioned, a temporary simplification of physical topology can be performed (e.g., hiding redundant links). However, our mechanism does not impact the resilience as links are just hidden during the allocation process not being deactivated. After converting physical and virtual graphs, a search space reduction is performed by aggregating capacity information of subtrees. This approach eliminates the need for a deep search as a capacity summary is available at each tree element. In brief, the main contributions of this work are 4-fold†:

- a mechanism to convert VI and network datacenter graphs in trees,
- an algorithm to reduce the search space by grouping subtrees information,
- an online tree-based algorithm to allocate VIs,
- experimental results with common cloud virtual topologies (hierarchical, multitiered, and VPCs) are discussed.

This paper is organized as follows: Section 2 formalizes the VIs allocation problem while Section 3 reviews the related work. Section 4 describes the proposed solution, discussing the conversion of virtual and physical graphs to trees, and explaining the proposed algorithm. The experimental analysis is discussed in Section 5, and Section 6 presents the conclusions and perspectives.

2 | VI ALLOCATION WITH VIRTUAL MACHINE AND NETWORK REQUIREMENTS

Recent innovations on network management and architectures have motivated the composition of private and isolated virtual topologies on cloud datacenters.[23] Currently, virtual networks with quality-of-service requirements can be dynamically

*Following the literature, mapping, allocation, and embedding have the same meaning is this article.

†An earlier version of this paper circulated in Portuguese only.[25]

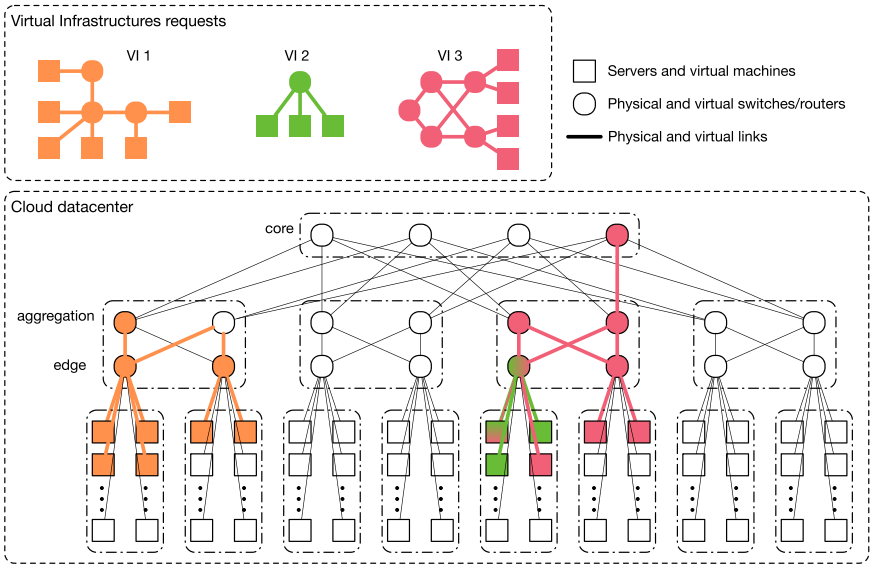


FIGURE 1 Example of a tree-based cloud datacenter hosting 3 virtual infrastructure (VI) requests with specific network topology configurations

composed to interconnect VMs. In cloud management frameworks, network provisioning is typically achieved via software-defined network (SDN) techniques[26] combined with traditional network virtualization mechanisms (e.g., virtual local area networks and multiprotocol label switching).[23] In addition, SDN controllers can be virtualized for sharing bandwidth, topology, and forwarding tables. Following this line, network function virtualization is in early stages of deployment and is pointed out as a promising approach for creating virtual network resources (e.g., switch, router, firewall, and network address translation).[27]

Through virtualization, cloud providers have abstracted their network datacenter resources to host isolated virtual entities belonging to multiple tenants. In this scenario, a single physical node (switch or router) can host multiple virtual entities (n:1 relationship), while a virtual link that connects 2 virtual nodes is mapped onto the path that connects the physical nodes hosting the virtual ones (n:m relationship).[10] Figure 1 depicts a scenario in which 3 VIs are allocated atop a cloud datacenter. Each tenant requested a VI with specific network topology, while datacenter is composed of servers interconnected by a tree-based topology[22] (in short, servers are interconnected by 3 layers of switches—core, aggregation, and edge). This example shows an arbitrary mapping solution to indicate that physical servers can host more than one virtual node, and similarly, physical paths can also host more than one virtual link (VI requests 2 and 3 share a subset of physical resources). Each virtual resource requires an amount of processing, storage, and networking (represented by virtual links) capacities that are provisioned by a cloud provider. It represents the SLA establishment between providers and tenants.

A cloud management framework relies on allocation algorithms to efficiently identify physical resources for hosting VIs. Usually, there are constraints that must be satisfied during the allocation process to guarantee that the physical infrastructure is capable of provisioning the requested virtual resources.[8] Indeed, the problem of cloud datacenter resources allocation to host VIs is a difficult one, both because it is computational expensive and on account of a wide range of constraints originating from different tenants and providers. In fact, the allocation cannot be randomly performed because of goals that must be achieved. For instance, providers want to allocate VIs efficiently, maximizing the acceptance rate and simultaneously minimizing the number of physical resources involved. Moreover, VIs can be placed anywhere atop a datacenter, but the allocation algorithm should reduce the spreading of virtual resources and the fragmentation of the physical datacenter, reducing the number of active resources.

On IaaS providers, the number of servers employed for composing a datacenter appears as a challenging aspect.[5] Moreover, VI allocation problem is exacerbated by the multicriteria constraints that must be satisfied. A VM may require a specific configuration of virtual CPUs, memory, and storage while virtual switches (or even routers) have another set of configuration such as size of flow table, memory, and processing capacity. Similarly, the physical topology must support the bandwidth requirements and resilience level required by the VI. Even a simple packet forwarding between servers requires a virtual link allocation for accomplishing the SLA. In short, all paths between physical resources are candidates for hosting a virtual link, as source and destination hosts are unknown in advance.

Formally, the problem of allocating VIs over physical infrastructures can be described as an extension of the VNE problem [11,12,28]: graph $G = (N, E, C)$ denotes a physical infrastructure where N is the set of physical nodes (servers, switches, and routers), E is the set of physical links, and each server or link is associated with a capacity vector C that indicates the available capacity. Similarly, graph $G_v = (N_v, E_v, C_v)$ denotes a VI request, where N_v and E_v represent the set of virtual nodes (VMs, switches, and routers) and virtual links, respectively, and C_v is the capacity vector of a virtual node or link. Therefore, the mapping of VIs on the physical resources

is described as $M{:}G_v \rightarrow (M_N, M_P)$, where $M_N{:}n^i \in N_v \rightarrow n \in N$ is a virtual to physical node mapping, and $M_P{:}e^i \in E_v \rightarrow p \in P$ is the mapping of virtual links over a physical path of P (a set of all physical paths). Each G_v is allocated considering the residual capacity of graph G. The residual capacity of a physical resource is the difference between the original and already allocated capacity. Finally, a mapping is considered a valid solution if for all virtual nodes and links, the physical resource hosting the virtual ones has enough capacity for provisioning the requested configuration, or in other words, $\forall n^i \in N_v$, $C_v(n^i) \leqslant C(M_N(n^i))$ and $\forall e^i \in E_v, C_v(e^i) \leqslant \min(C(l), \forall l \in P)$.

In addition to delivering the requested capacities for tenants, a cloud provider must reduce the number of active resources (servers, switches, and links). Moreover, for communication-intensive applications, the distance between virtual resources is a critical factor.[23] In this sense, we used the fragmentation metric[15] for defining the objective function of a cloud provider. Fragmentation quantifies the percentage of active resources of a datacenter that are hosting VIs, figured by dividing the number of active resources by the total number of available resources. In this context, finding optimal solutions for graph-embedding problems with constraints in nodes and edges is a well-known NP-hard problem. A VI allocation mechanism must simultaneously identify appropriated physical equipments and paths for hosting VMs, switches, and links. In this case, searching by optimal solutions is impracticable as cloud tenants seek an immediate response for VI provisioning requests.

An intuitive approach is to perform vertices allocation and subsequently virtual network composition. If performed on a naive way, a 2-phase mechanism may induces datacenter underutilization and negatively impact on hosted applications (a virtual link can spam a long physical path increasing end-to-end latency). An initial vertices allocation mechanism to place VMs without considering edges and networking equipments (switches) can select spread resources placed on different parts of a cloud datacenter (e.g., blades, racks, zones, and regions). Although second-phase algorithms can find an optimal network composition, the final mapping is still suboptimal for tenants and providers. Tenants may be subject to high latency (because of network hops) on internal VMs communication, which can decrease the hosted application performance.[29] For providers, composing long paths for hosting virtual links requires the activation of multiples switches along the path. Even using SDN technologies, management of tenants flows is complex as multiple flow-table entries are required for isolating the VIs.[30] These challenges lead to application of allocation heuristics, as described in the present paper: a joint allocation of VM and network requirements can decrease the spread of virtual resources atop a cloud datacenter.

3 | RELATED WORK

Specialized literature comprises several proposals to allocate physical resources to host virtual entities. Most studies focus on the VNE problem[8] while some aims VM placement on cloud providers. In short, VNE allocates a set of virtual routers (connected by virtual links) on physical topologies.[10] There are different proposals on solving this graph-embedding problem, including isomorphism-based identification, path-splitting methods, multicommodity flow modeling, and heuristics built on substrate characteristics. In addition, they have their own objectives and metrics, for example, maximizing resource use, minimizing the maximum link load, proposing fault-tolerant maps, and allocating virtual resources across multiple domains.

In VNE-related literature, Yu et al[11] introduced virtual links splitting over multiple physical paths combined with periodic datacenter rebalancing. The proposal modeled the VNE as a multicommodity flow problem and highlighted in their evaluation that the splitting of virtual links can maximize revenue and minimize substrate resource usage. Although their goal was not the simultaneous allocation of VMs (as proposed in the present work), the metric used to identify the potentially complex candidates was based on node connectivity degree and served as a base for comparison with the algorithm described in Section 4. Lischka and Karl[28] proposed an exhaustive search by isomorphic graphs. The algorithm is an extension of the vflib graph matching algorithm.[31] In this approach, nodes and links are simultaneously analyzed during the allocation process, and by limiting the number of physical hops of a virtual link, the proposal reduced the search space. However, experimental analysis indicated allocation time in minutes, a waiting time that is unacceptable for cloud tenants expectations (our proposal achieves a stable performance with short allocation time, as discussed in Section 5).

Some approaches have proposed strategies to exclusively allocate VMs atop virtualized datacenters. They commonly explore linear programming or heuristics to dynamically reserve and place VMs on public cloud providers, differentiating the objective function, as dynamic allocation[19] with VMs migration,[17] local capacity extension by borrowing resources from public cloud,[18] optimal welfare,[32] and power savings.[33] However, the problem is partially solved as virtual networks are not considered or mapped in different phases. Some heuristics perform VM mapping and lately a shortest-path search. By selecting VMs, switches, and links in different phases, the algorithms perform the allocation in a 2-step process, first allocating the nodes and then allocating the links, solving a shortest path or multicommodity flow problem. A 2-step approach leads to suboptimal solutions, increasing the latency between virtual elements and unbalancing the network datacenter.[29] Aiming an efficient map, an allocation algorithm must consider nodes and links with the same importance, allocating both simultaneously.[12] The algorithm proposed in Section 4 is aligned with this expectation and performs a joint allocation of network and VM resources.

Chowdhury et al[12] introduced the combined allocation of virtual routers and links, that is, both are mapped in a single step by the algorithm. This innovation led to the inclusion of VMs in the problem formulation and the design of the algorithm proposed in this paper. With link and node constraints, the authors formulated the problem as a mixed

integer program, and the integer constraints were relaxed to obtain a linear programming. Samuel et al[34] extended the formulation to consider the allocation of VIs across multiple administrative domains that compose the physical substrate. In complement, Yeow et al[35] investigated the VI allocation considering mechanisms to pool backup nodes so as to achieve the desired level of reliability together with resource allocation. However, all proposals tried to relax some constraints, later applying heuristics for verifying if mappings are applicable. Our proposal achieves a combined allocation of links and nodes on cloud datacenters using well-known tree-based algorithms (an updated version of Chowdhury et al mechanism[36] is discussed and compared with our proposal in Section 5.).

The diversity of virtual topologies has not been the target of most work, which motivated the proposal of Luizelli et al.[20,21] The authors analyzed the impact of different physical topologies on VNE quality, mainly focusing on rejection rate. In parallel, Butt et al[37] investigated a topology-aware mechanism to reconfigure and allocate initially rejected requests. Following this line, Wang et al[14] proposed a solution based on tree topologies. However, the approach has not considered the combined allocation of communication resources and VMs, and virtual requests were limited to tree-based topology. Our proposal relies on a tree-based algorithm to perform the allocation but is not limited to this topology subset (as discussed in Section 4.1). In complement, Cheng et al[13] and Gong et al[38] did not limit the problem formulation for only tree-based topologies and explored their characteristics to identify the critical nodes (by degree of connectivity). However, both formulations discussed a new perspective for VNE problem, without considering VM placement.

Although our proposal focus is the allocation of VIs, some partial algorithms and metrics for comparison are derived from VNE and were considered as guidelines for developing the formulation and experimental analysis. Thus, Section 4 proposes an approach to VIs allocation applying concepts related to trees to decrease the search space and to find an efficient mapping solution. Experimental analysis performed in Section 5 compares our proposal with a well-accepted solution[36] commonly used as a baseline for joint resources allocation. The algorithm was selected to be compared as it has outperformed most VNE heuristics, serving as base for mixed integer linear programming approaches.

4 | A NEW APPROACH FOR VIRTUAL INFRASTRUCTURE ALLOCATION

This paper proposes an algorithm termed VITreeM (VI tree–based mapping) for the online allocation of physical resources to host VIs. VITreeM has as premise that both graphs, VI request and physical topology, are in the form of trees. In graph theory, a tree is an undirected graph G that satisfies the following conditions: (1) G is connected, that is, there is a path between any 2 vertices of G, and (2) G has no cycles, specifically, there is only one path between any 2 vertices of G. This premise is motivated by the current use of tree-based topologies in datacenters[22] and by the promising application of polynomial-time algorithms in allocation process.

However, it is known that the topologies of physical and VIs are not necessarily specified as trees,[21,39] usually breaking the second condition by the existence of cycles.[20] To overcome this limitation, VITreeM introduces a strategy to convert any graphs on trees. Afterwards, the mechanism performs a joint allocation of VMs and network resources considering both aspects simultaneously. In summary, VITreeM is composed of (1) an algorithm to convert graphs in trees keeping the original connectivity requirement between vertices; (2) an approach to define the starting nodes (the roots of both VI and datacenter infrastructure trees) for the allocation; (3) a grouping strategy to simplify the representation of subtrees and to reduce the search space; and (4) a tree-based allocation mechanism guided by the objective and constraints defined in Section 2.

4.1 | Converting virtual infrastructures and network datacenter graphs to trees

Ahead performing an allocation, VITreeM verifies if physical and virtual graphs are in the form of trees. As the original graph representing a VI or a datacenter must be connected, the infringement on tree properties occurs in the existence of cycles. In this case, VITreeM converts graphs to trees by removing cycles based on maximum spanning tree (MST). A spanning tree of a graph $G = (N, E, C)$ is a subgraph $T = (N^T, E^T, C^T)$ that has all properties of a tree and contains all nodes of G, in other words, $N^T = N$ and $E^T \subseteq E$. In weighted graphs, the MST of G corresponds to the heaviest spanning tree.

For obtaining the MST of a graph, VITreeM applies an adaptation of the Kruskal algorithm originally developed for the minimum spanning tree searching.[40] Basically, the order of the edges evaluation was changed by replacing the nondescending order by a nonascending order. When multiple MSTs are found, one is arbitrary selected as they are all equivalents.

On network datacenter graphs, edges not present in the MST are ignored in mapping process (they are temporary hidden). However, among all spanning trees obtained, the MST has the greatest chances of success in hosting because of implicit maximization of the weights in the selected communication paths. Figure 2 shows an example of a network

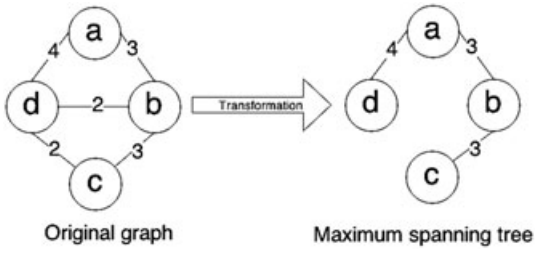


FIGURE 2 An example of a maximum spanning tree for a network datacenter graph

datacenter graph and the corresponding MST. Throughout the conversion, edges (d, b), (d, c) were removed to obtain a MST.

However, removing edges that are not included in the MST ($T_v = (N_v^T, E_v^T, C_v^T)$) of a VI request (a graph $G_v = (N_v, E_v, C_v)$) is not acceptable as it disarticulates the virtual topology structure. To overcome this limitation, we added the weight of every edge $(u, v) \in E_v - E_v^T$ to the edges belonging to the path between u and v of T_v. Figure 3 illustrates the conversion process applied to an IV: From the original graph, VITreeM obtains the MST by removing the edges (d, b), $(d, c) \in E_v$. The weights from these edges are added to the MST between the source and destination nodes. For example, the original edge (d, b) is replaced on the MST by *d-a-b*, and consequently, the edge's weight ($(d, b) \in E_v$) is added in the edges $(d, a), (a, b) \in E_v^T$. Finally, the edge $(d, b) \in E_v$ is now represented by path *d-a-b* in the MST, maintaining the connectivity between virtual nodes as originally requested.

4.2 | Setting the starting node for allocation

The essence of VITreeM comprehends a depth search comparing virtual and physical trees, followed by a breadth inspection when needed. Thus, VITreeM needs to identify a node of each tree to start the search for an allocation. As discussed in previous work, [11,38] the starting search point influences the distribution of virtual resources atop the physical infrastructure as it guides how the algorithm will conduct the search for the candidates. In approaches were virtual links are hosted on physical paths with limited length, setting the starting node is even more complicated as it can reduce the candidates to a local search.[28] When this restriction is not applied, the starting node can impact on final network load.[13,34] Our proposal uses 2 strategies for starting node selection: identification of center of a tree and search for the node with largest local resource capacity (LRC)[11] (they are compared in Section 5) .

In graph theory, the eccentricity of a node n from a given tree is defined as the greatest distance between n and any other vertex. At the same time, the center of a tree is a node with the smallest eccentricity and can be achieved by iteratively removing tree leaves, until only one or two nodes remain.[41] Figure 4A illustrates the achievement of the center following this strategy. Leaves (red) are iteratively removed until the identification of node c that is the center of the tree (in green).

The LRC metric is used to identify potential complex candidates, given by the product of the node u weight and the sum of all the edge's weight connected with u (formally, $LRC = C(u) * \sum C(u, v), \forall v \in N^T \wedge (u, v) \in E^T$).[11] The higher the value of the node's LRC, the lower the probability to allocate it on the leaves of a tree because of its greater demand for resources. Figure 4B illustrates the LRC calculation for all nodes on a given tree and, finally, the selection of the node d due to its higher LRC.

4.3 | Grouping subtrees in boxes

One of the main challenges identified in VI allocation is the resulting combinatorial explosion due to elevated number of candidates available in cloud datacenters. It is well-known that the abstraction of physical candidates through grouping techniques can reduce the number of operations required to verify if a mapping is feasible.[42] VITreeM uses a heuristic, termed boxes, to group information on branches of a tree and consequently to reduce the search space. A box contains a tuple with 3 capacity values (links, servers, and routers) obtained by the sum of capacities in subtrees rooted at a given

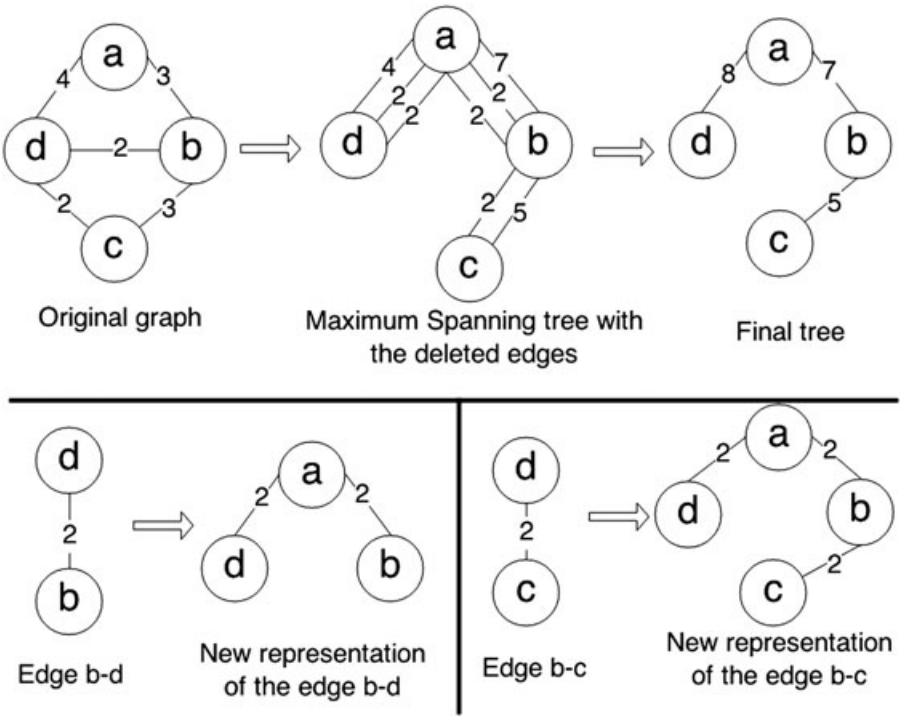
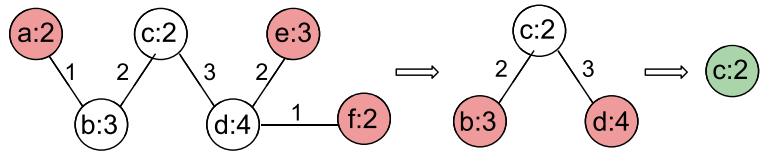
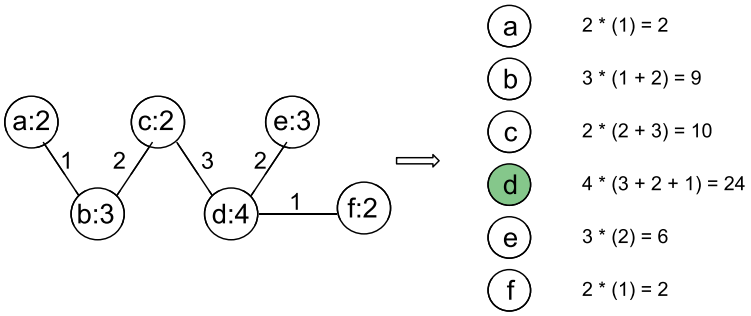


FIGURE 3 The conversion of a virtual infrastructure (VI) graph in a maximum spanning tree



(a) Definition of the central node of a tree by iteratively removing leaves.



(b) Definition of the LRC [11].

FIGURE 4 Identification of the allocation starting node. LRC indicates local resource capacity

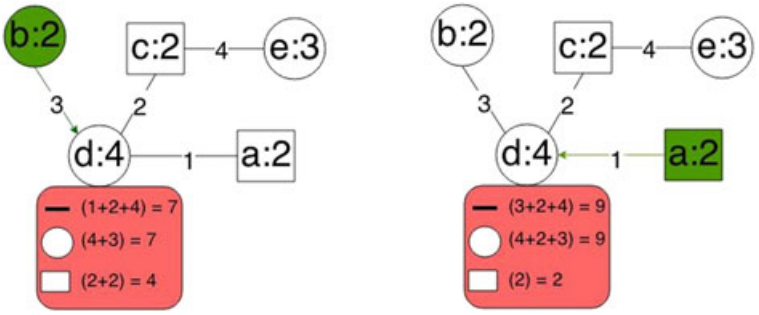


FIGURE 5 Example of capabilities grouped on boxes

node. Moreover, a box content may change depending on observing reference. This dynamic representation is used to calculate hosting physical paths on multiple directions. With boxes representation, VITreeM has a notion of all available resources in a subtree rooted by a particular node without the need for a deep investigation on the tree.

An example of grouping boxes is presented in Figure 5. Routers, servers, and links are represented by circles, rectangles, and solid lines, respectively. Capacity boxes are highlighted in red while observing nodes are represented by green vertices. The box of node d viewed from the node b, noted as ($box(b \rightarrow d)$), contains different values from those observed from node a, ($box(a \rightarrow d)$), because they form different subtrees. VITreeM has a view of the amount of resources that are present in a subtree and quantifies the potential for mapping without exploring all the search space elements.

4.4 | Allocating physical resources to host VIs

VITreeM is a combination of all heuristics described above (the steps are performed in the order they were described). With focus on online requests, this recursive algorithm allocates VIs individually (VI requests arrive in discrete time intervals). Pseudocode from Algorithm 1 resumes VITreeM, which receives as input: (1) a graph $G = (N, E, C)$ representing the physical infrastructure; (2) a graph $G_v = (N_v, E_v, C_v)$ denoting the original VI request; (3) a central node n_G from tree T, where $T = graph\_to\_tree(G)$; (4) a central node n^v from tree T_v, where $T_v = graph\_to\_tree(G_v)$; (5) a tree T; (6) a tree T_v; and (7) a map $M = (M_N, M_P)$ initially empty. If a solution is found, the algorithm provides a mapping M of G_v on G, indicating which physical resources were compromised to host the VI request.

Initially, the algorithm performs an attempt mapping of T_v, rooted by node n^v, on some subtree of T rooted by neighboring nodes of n_G (lines 1-4). For each subtree rooted by neighbors u of n_G, VITreeM compares the box values of n^v with the box information of u. The *previous*() function indicates the last allocated candidate or the node itself when the algorithm starts. The boxes comparison is one of the key points of the algorithm, since it reduces the search space, avoiding unnecessary checking in deep nodes when it is detected that the subtree n^v is not supported by the subtree of u. If the subtree has enough resources, the algorithm is executed recursively continuing the mapping (line 3).

If a valid mapping is not found in the process described above, VITreeM tries to map the virtual node n^v on the physical node n_G (lines 5-8). If possible, it carries out the mapping

of the subtrees rooted by n^v on the subtrees rooted by neighboring nodes of n_G (lines 9-18). To do so, it creates a set (identified by *candidates*) with all possible pairs between neighbors of n_G and neighbors of n^v. The order in which the pairs are selected follows pre-established criteria, such as the best fit (nodes are sorted in ascending residual capacity order [the difference between physical and virtual boxes]) and worst fit (which follows a descending order).

Algorithm 1: *VITreeM* - a tree-based algorithm to allocate VIs.

Data: $G = (N, E, C)$; $G_v = (N_v, E_v, C_v)$; $n_G \in T$; $n^v \in T_v$; T; T_v; $M = (M_N, M_P)$;
Result: map M of G_v on G, if possible

1 **for** $\forall(u, n_G) \in T$ *such that* $u \in T$ **do**
2 **if** $box(n_G \rightarrow u) \geq box(previous(n^v) \rightarrow n^v)$ **then**
3 **if** $VITreeM(G, G_v, u, n^v, T, T_v, M)$ **then**
4 **return** $(true, M)$;
5 **if** $C(n_G) < C_v(n^v)$ **then**
6 **return** $(false, null)$;
7 $M_N(n^v) = n_G$;
8 $C(n_G) = C(n_G) - C_v(n^v)$;
9 $candidates = \{(u \in N, v \in N_v), \forall(n_G, u) \in E \wedge (n^v, v) \in E_v\}$;
10 **while** $\exists v \in N_v \wedge v \notin M_N \wedge \#candidates > 0$ **do**
11 $(u, v) = next\_pair(candidates)$;
12 $candidates = candidates - \{(u, v)\}$;
13 **if** $C((n_G, u)) \geq C_v((n^v, v)) \wedge box(n_G \rightarrow u) \geq box(n^v \rightarrow v)$ **then**
14 $C((n_G, u)) = C((n_G, u)) - C_v((n^v, v))$;
15 $M_P((n^v, v)) = M_P((n^v, v)) \cup (n_G, u)$;
16 **if** ***not*** $VITreeM(G, G_v, u, v, T, T_v, M)$ **then**
17 $C((n_G, u)) = C((n_G, u)) + C_v((n^v, v))$;
18 $M_P((n^v, v)) = M_P((n^v, v)) - (n_G, u)$;
19 **if** $\forall(n^v, v) \in E_v \wedge v \in M_N$ **then**
20 **return** $(true, M)$;
21 **else**
22 remove all the maps from this point;
23 **return** $(false, null)$;

Finally, VITreeM verifies the existence of a valid mapping to n^v neighbors; otherwise, all the mapping performed from the current call of the algorithm is undone. This algorithm has an asymptotic complexity of $O(n^2\log(n))$ to the best case, in which the allocation is performed in the first attempt. On the other hand, the complexity is $O(2^n\log(n))$ for the worst case, where VITreeM verifies all the mapping possibilities when there is enough capacity in all singular boxes of the physical graph to allocate the greatest node in the virtual graph. In this situation, it is possible that the entire virtual graph does not fit in the physical graph, and the algorithm will search for all possibilities. Although the theoretical complexity in the worst case is exponential, the impact is smoothed by search space reduction introduced by grouping subtrees information in boxes.

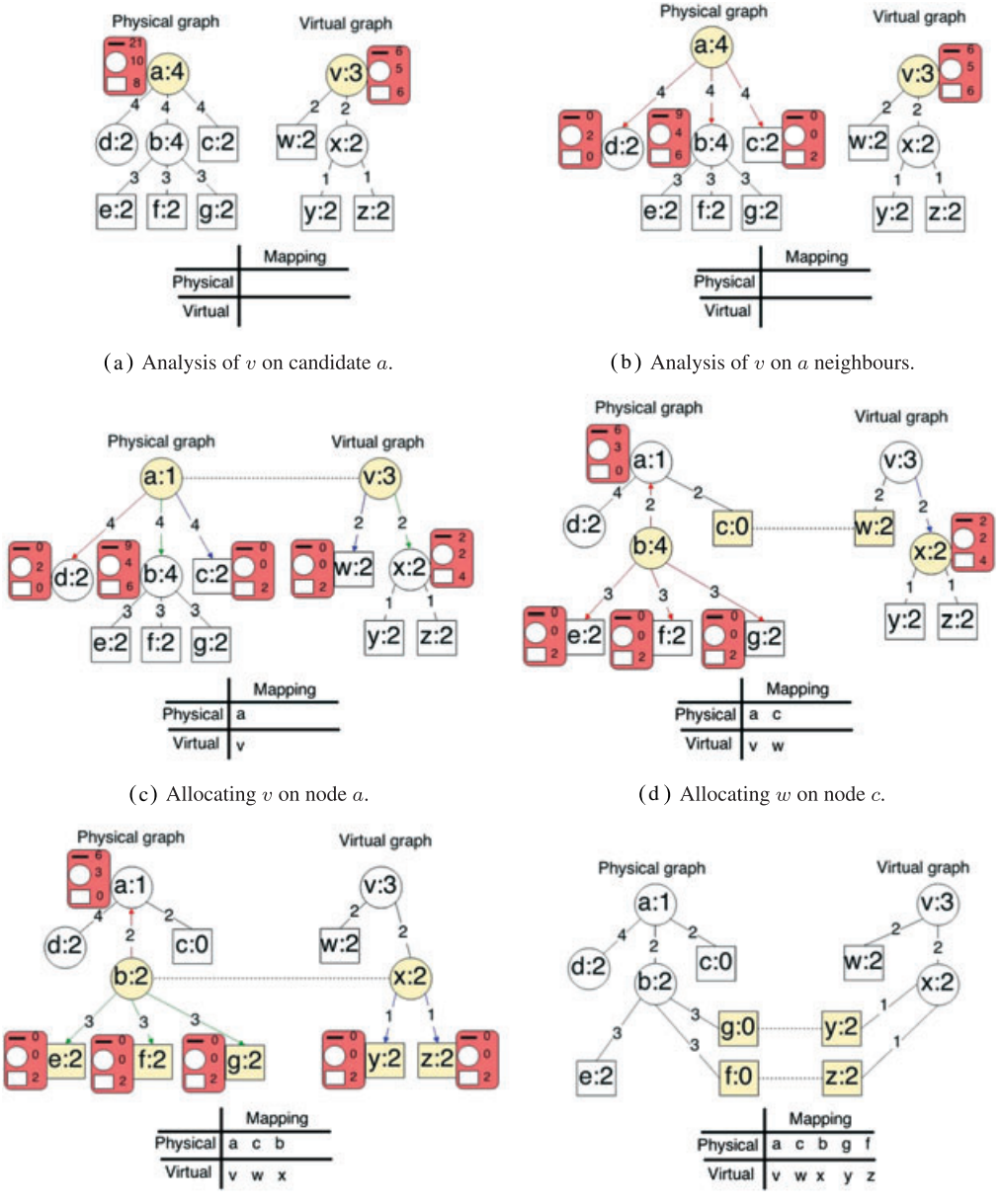
To illustrate the VITreeM applicability, Figure 6 exemplifies a tree-based search for a VI request and a physical infrastructure (both trees are in MST form). Initially, Figure 6A represents the analysis of node $a \in N^T$ (physical tree $T = (N^T, E^T, C^T)$) and of node $v \in N_v^T$ from virtual tree $T_v = (N_v^T, E_v^T, C_v^T)$. By comparing the values of each item of both boxes, we observe the possibility of allocating T_v on T, as the values of node a are greater than the values of node v. The second step (Figure 6B) seeks a neighbor node of a capable of supporting the box of node v (lines 1-4 of Algorithm 1). By only comparing the boxes, we conclude that there is no node in the subtrees (among a neighbors) with enough resources to support v.

As illustrated in Figure 6C, it is not possible to continue the allocation by a neighbors. Thus, v is mapped on a (lines 7 and 8 of Algorithm 1). Following this line, the node $w \in N_v^T$ is mapped on $c \in N^T$ as any neighbor of c is capable of hosting w (Figure 6D). The same situation occurs with the node $x \in N_v^T$: It is not possible to move forward on $b \in N^T$ neighbors, since they do not have sufficient capacity to allocate it. Consequently, x is mapped on b, leaving only its neighbors e,f,$g \in N^T$ to be allocated (lines 9-18 of Algorithm 1). Figure 6E depicts the available physical candidates ($e, f, g \in N^T$) while Figure 6F shows the final state of the allocation.

5 | EXPERIMENTAL ANALYSIS

The experimental analysis compares VITreeM with a well-known mechanism[36] proposed to VNE with coordinated node and link mapping that outperformed existing approaches from specialized literature. To allocate a virtual network request, the VNE problem was formulated by Chowdhury et al as a mixed integer program with relaxed constraints. The authors relaxed the domain of binary variables (representing the resources map) to obtain a linear program. The initial solution with relaxed constraints is used to devise deterministic (DVINE) and randomized (RVINE) algorithms. For performing the allocation of VIs, the mechanism was lightly adapted for considering the existence of virtual routers and VMs (the vertices were decomposed in 2 subsets with specific constraints). DVINE and RVINE are based on a distance parameter that limits the number of physical candidates for hosting a virtual one (it is similar to the starting points discussed in Section 4.2). We set this parameter with different values defining a percentage of physical candidates applicable for each virtual element. DVINE and RVINE were solved using CPLEX optimizer with its default configuration.[‡] As RVINE results are similar or worse than DVINE results for

[‡]CPLEX Optimizer, available at http://www.ibm.com/software/commerce/optimization/cplex-optimizer/



(a) Analysis of v on candidate a.

(b) Analysis of v on a neighbours.

(c) Allocating v on node a.

(d) Allocating w on node c.

(e) Analysis of nodes y and z.

(f) Allocating nodes y and z on g and f.

FIGURE 6 Example of the tree-based allocation using VITreeM

the experimental scenarios analyzed, we opted to not discuss RVINE results in this section.[§]

The VITreeM algorithm was implemented in *C++* (compiled by *g++4.8* with the *-O2* optimization parameter).[¶] For covering all input parameters, 4 configurations of VITreeM are compared: the starting node selection (LRC or center) and the order in which the pairs of candidates are evaluated (best fit or worst fit). Basically, the former compares methods to identify the roots of physical and virtual trees, while the latter guides tree searching. As discussed in Section 4, the definition of these parameters impacts the performance of allocation algorithms in general. Table 1 resumes the algorithms compared in experimental analysis.[‖]

Following the work of De S Cavalcanti et al,[15] physical network datacenter was based on Cisco 3-layer datacenter model,[22] as exemplified by Figure 1. A simplification of this datacenter topology was defined as (1) access layer, composed of 4 racks, each containing 12 servers and 1 top-of-rack switch; (2) aggregation layer with 4 switches; and (3) core layer with 2 switches. Based on this configuration, we prepared a set of scenarios representing providers with different infrastructures. Table 2 summarizes the total number of nodes

[§]On worse scenarios, RVINE allocated approximately 32% less requests than DVINE. Although an extensive analysis on RVINE is out of the scope of this paper, we believe it is related with datacenter topology and dense virtual requests.

[¶]The code is available at https://bitbucket.org/ramoncs/VITreeM

[‖]For the investigated scenarios, increasing the maximum distance in both RVINE and DVINE did not improve the algorithms' performance and just affected the execution time.

TABLE 1 Algorithms and configurations compared in experimental analysis

Algorithm	Parameters and Configuration	Label
VITreeM	LRC and ascending capacity order	*LRC | BF*
VITreeM	LRC and descending capacity order	*LRC | WF*
VITreeM	Center node and ascending capacity order	*CENTER | BF*
VITreeM	Center node and descending capacity order	*CENTER | WF*
DVINE	Up to 30% physical candidates per virtual node (maximum distance = 1)	*DVINE | 1*
DVINE	Up to 50% physical candidates per virtual node (maximum distance = 2)	*DVINE | 2*
DVINE	Up to 70% physical candidates per virtual node (maximum distance = 3)	*DVINE | 3*

Abbreviations: BF, best fit; DVINE, deterministic; LRC, local resource capacity; WF, worst fit.

TABLE 2 Composition of cloud datacenters based on Cisco 3-layer datacenter model[22]

Scenario	No. of Datacenters	No. of Servers	No. of Switches	Total
SC 1	1	48	10	58
SC 2	3	144	30	174
SC 3	6	288	60	348
SC 4	9	432	90	522
SC 5	12	576	120	696
SC 6	18	864	180	1044
SC 7	24	1152	240	1392

for each scenario, varying from small-scale datacenters (SC 1) to configurations with 1392 physical resources.

For analyzing VITreeM applicability on IaaS providers, 2 VI topologies commonly used in cloud environment were selected: multitiered and VPCs. Later, a set of hierarchical VI requests representing virtual research testbeds is analyzed.

- *Multitiered VI requests:* A large fraction of IaaS tenants request and organize their VMs following a *n*-layers topology.[24] Usually, the topology is composed of a load balancer in charge of distributing requests for a set of web servers, which represents the hosted application logic. Eventually, web servers search data from database servers. A possible composition of a multitiered VI request is represented by VI 1 from Figure 1. In this example, each layer has a dedicated virtual switch for individually requesting network quality-of-service parameters. For creating multitiered virtual requests, in addition to virtual switches, each request was composed of one load balancer, and a set of web and database servers. The number of VMs for representing web servers and databases is uniformly distributed between 50 and 100 (for each subset), while communication between VMs and switches is represented by virtual links request.
- *Virtual private clouds:* Recently, Amazon EC2 introduced the dynamic provision of VPCs. [**] Each VPC is composed of a subset of access point rules (represented as virtual routers) and a set of VMs (as depicted by VI 2 from Figure 1). In a VPC, VMs are directly connected to a virtual router, composing a private local network that is managed by the tenant. On experimental analysis, the number of VMs composing a VPC follows a uniform distribution between 50 and 200 elements. Each subset of VMs is connected to a virtual router represented by a virtual link request.
- *Hierarchical VI requests:* As virtualization has appeared as a promising technology for evolving the internet, one can compose a virtual testbed based on virtual switches/routers for testing new protocols and management tools. This scenario is represented by a hierarchical VI request (as depicted by VI 3 from Figure 1). Virtual machines are interconnected by a set of switches organized in layers (similarly to a tree-based topology[39]). For creating an experimental set of requests, the number of layers is uniformly distributed between 2 and 7. Each layer contains 2 to 5 switches except for the last layer, which has between 5 and 10 VMs, all following a uniform distribution.

Following previous work,[15,43] virtual requested capacity is defined as a fraction of total physical capacity: each virtual router, VM, or virtual link consumes 1% to 10% of a physical resource (following a uniform distribution). A set of 100 virtual requests is generated and submitted for each physical scenario described in Table 2. Arriving times are uniformly distributed up to 150 discrete intervals (a VI remains active for 30 intervals at most), and the results show sample means with 95% confidence intervals (given by error bars on graphs or denoted by ± on tables). All experiments were performed on an Intel Core i5 with 16-GB RAM.

5.1 | Metrics for analysis

The specialized literature uses a set of common metrics for analyzing the performance and quality of embedding

[**] Available at https://aws.amazon.com/vpc/

approaches: fragmentation, allocation time, revenue, acceptance rate, and load of physical resources (nodes and links).

1. Datacenter *fragmentation* indicates the percentage of active resources that are hosting VIs, figured out by dividing the number of active resources by the total number of available resources.
2. *Allocation revenue* gives an insight of how much a cloud provider will gain by accepting a VI request. It is calculated by summing up the requested virtual capacity of a specific VI (in other words, it is given by the sum of vertices and edges weights from a virtual graph).
3. The *mean runtime to allocate* a VI request.
4. *Acceptance rate* quantifies the percentage of successfully allocated VIs. At a given interval, it is calculated as a ratio of allocated VIs to total number of requests analyzed so far.
5. *Load of physical nodes and links* represent the percentage of resources that are used. It is computed by the division of the sum of the allocated capacities by the total capacity, allowing the identification of physical resources committed in the VIs allocation (and bottlenecks).

Substrate fragmentation and datacenter load (nodes and links) are related with network performance. When datacenter is under-provisioned or load is balanced, hosted applications are not impacted by communication bottlenecks. Ideally, an allocation mechanism should decrease fragmentation, keeping virtual resources proximity, while controlling substrate load to avoid bottlenecks. Acceptance rate is influenced by both load and fragmentation metrics. In short, a well-balanced datacenter should be capable of hosting more requests faced to a fragmented one. An analysis of allocation revenue combined with acceptance rate indicates if the mechanism prioritize weighted requests (a higher revenue), or light, with a low revenue. Finally, algorithm's runtime represents how long a requesting tenant should wait for the cloud management framework processing.

5.2 | Experimental results

Figures 7, 8, and 9 present the results for multitiered, VPC, and hierarchical topologies, respectively, executed on scenarios SC 1, SC 2, SC 3, and SC 4. Metrics obtained by executing VITreeM on large-scale scenarios (SC 5, SC 6, and SC 7) are shown in Tables 3, 4, and 5. For performing a comparison between allocation mechanisms and configurations, each metric is depicted by an individual graph, in which results are grouped per physical scenario (from Table 2).

5.2.1 | Multitiered VI requests

First, we analyze the acceptance rate of multitiered VI requests, depicted by Figure 7A and Table 3. For most physical scenarios, VITreeM with the best-fit candidates ordering accepted more requests than the DVINE and worst-fit ordering (except for SC 7 in which the worst-fit approach obtained equivalent results). The best-fit candidates selection tends to consolidate virtual resources on already activated servers as it only elect idle hosts when residual capacity is not enough for hosting the virtual request. This impact is observed on fragmentation (Figure 7C) and revenue (Figure 7D) graphics: Even accepting more requests and consequently obtaining a higher revenue, VITreeM achieves a datacenter fragmentation equivalent to DVINE.

Virtual machines composing multitiered VI requests are interconnected by a set of virtual links (as depicted by VI 1 from Figure 1), composing a virtual topology without communication cycles. In this sense, the graph to tree conversion proposed by VITreeM is not required. While DVINE has consistent nodes and links loads even scaling out the datacenter (Figure 7E and F), it was unable to use the remaining capacity for hosting more virtual requests. In fact, DVINE tends to spread virtual resources atop a datacenter faced to VITreeM, as highlighted by a combined revenue and fragmentation analysis: even allocating more virtual requests, VITreeM has a resulting fragmentation comparable with that of DVINE, indicating a datacenter consolidation.

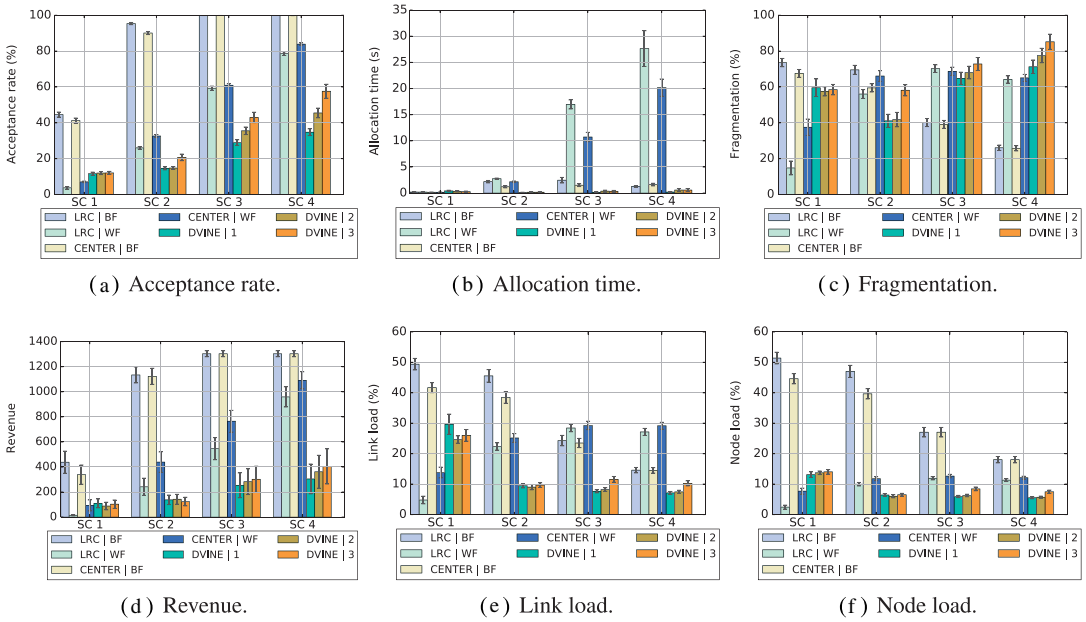
Specifically for large-scale scenarios (SC 5 to SC 7), VITreeM with the best-fit ordering achieved 100% of acceptance rate. For each scenario, the selection of a starting node based on LRC or central node of a tree resulted in equivalent metrics. Consistently, VITreeM with the worst-fit ordering used more physical network resources for hosting the same subset of virtual requests. For instance, on SC 7, both approaches obtained 100% of acceptance rate, although the worst fit used 3 times more networking resources for hosting a set of virtual requests with equivalent revenue.

Considering the allocation time, VITreeM with the best-fit ordering and central node searching has obtained results similar to those of DVINE (Figure 7B). Specifically for multitiered requests, the average allocation time of CENTER | BF configuration on a large-scale datacenter (SC 7) was approximately 19.28 seconds, while for small datacenters (SC 1 to SC 4), it was less than 4 seconds.

5.2.2 | VPC requests

Virtual private clouds–based virtual requests (exemplified by VI 2 from Figure 1) are composed of a set of VMs interconnected by a virtual router. Internal connectivity requirements between VMs are expressed as virtual links. Consequently, VPC has no cycles. However, different from multitiered requests, the number of virtual links connected to a single point is increased, composing a large-scale star topology.

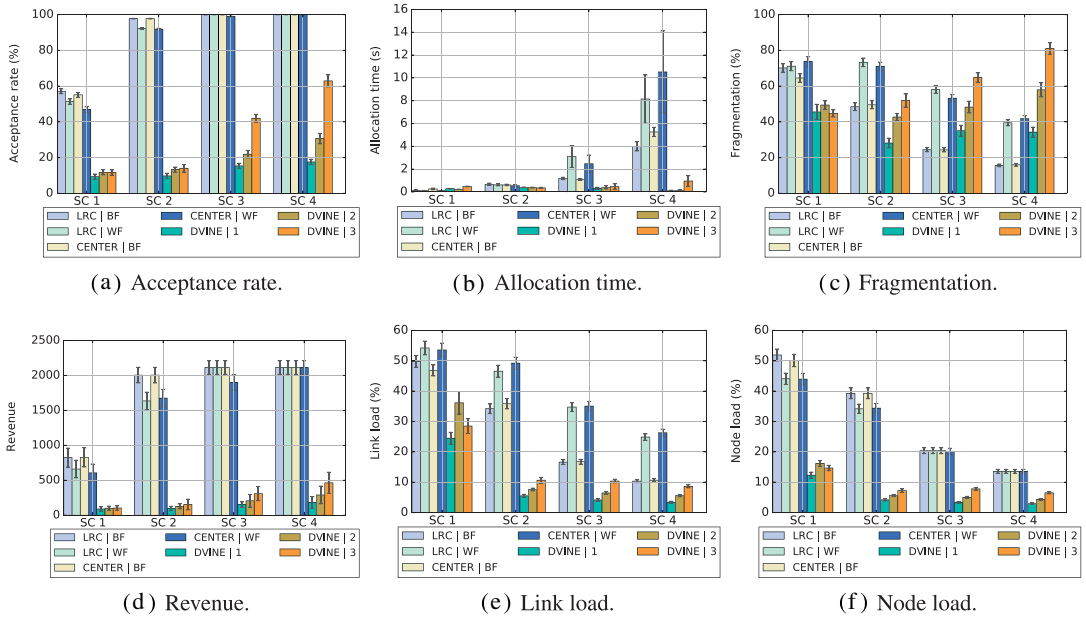
Results for VPC requests are presented in Figure 8 and Table 4. Even on small-scale datacenters (SC 2), VITreeM accepted almost all submitted requests, while DVINE allocated approximately 64% requests at most (SC 4, Figure 8A). For this scenario, the best-fit and worst-fit selection approaches obtained high acceptance rate independently of starting node selection.



(a) Acceptance rate. (b) Allocation time. (c) Fragmentation.

(d) Revenue. (e) Link load. (f) Node load.

FIGURE 7 Simulation results for multitiered virtual infrastructure requests allocated by VITreeM (LRC, CENTER, BF, and WF) and DVINE (1, 2, and 3) configurations (from Table 1). BF indicates best fit; DVINE, deterministic; LRC, local resource capacity; WF, worst fit



(a) Acceptance rate. (b) Allocation time. (c) Fragmentation.

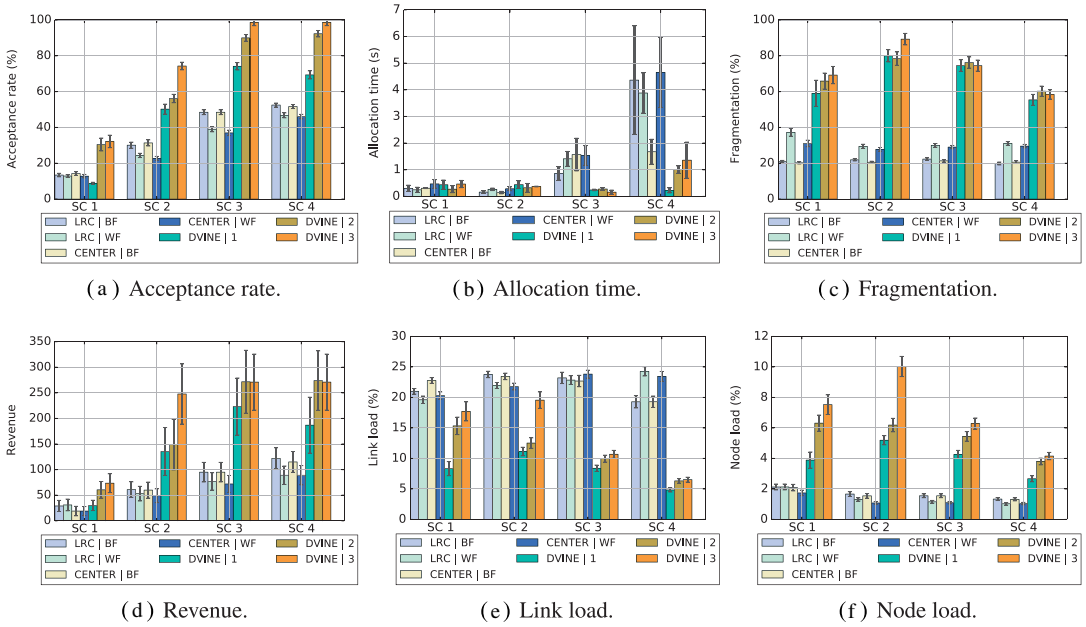
(d) Revenue. (e) Link load. (f) Node load.

FIGURE 8 Simulation results for VPC-based requests allocated by VITreeM (LRC, CENTER, BF, and WF) and DVINE (1, 2, and 3) configurations (from Table 1). BF indicates best fit; DVINE, deterministic; LRC, local resource capacity; VPC, virtual private cloud; WF, worst fit

Considering revenue and fragmentation metrics, DVINE obtained a low revenue (Figure 8D) even activating more links and servers on a datacenter (Figure 8C). Although nodes and links loads (Figure 8E and F) indicate that a considerable residual capacity is available for hosting virtual resources, DVINE opts for hosting a unique VM per server on its formulation.[36] Consequently, VPC requests submitted to DVINE are limited by the number of available resources. In practice, physical resource consolidation is not applied for online allocations (individual VI request).

For each scenario with 100% of acceptance rate, VITreeM configurations obtained equivalent numbers. Indeed, a considerable difference is observed on link load as the worst-fit approach tends to spread resources atop a datacenter requiring the provisioning of long paths for hosting virtual links.

The average allocation time of VITreeM for VPC-based requests atop a large-scale scenario (SC 7) was approximately 17.86 seconds using the best-fit ordering and central node approach. Although for VITreeM the number of physical candidates to host a VI is directly related to allocation time, an exponential growing is not observed on large-scale scenarios (SC 5 to SC 7), highlighting the candidates prune performed by grouping boxes.



(a) Acceptance rate. (b) Allocation time. (c) Fragmentation.

(d) Revenue. (e) Link load. (f) Node load.

FIGURE 9 Simulation results for hierarchical virtual infrastructure requests allocated by VITreeM (LRC, CENTER, BF, and WF) and DVINE (1, 2, and 3) configurations (from Table 1). BF indicates best fit; DVINE, deterministic; LRC, local resource capacity; WF, worst fit

TABLE 3 Results for multitiered VI requests submitted to scenarios 5, 6, and 7

	SC 5				SC 6				SC 7			
	LRC		CENTER		LRC		CENTER		LRC		CENTER	
	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF
Acceptance rate, %	100.0	84.92	100.0	90.28	100.0	99.31	100.0	97.14	100.0	100.0	100.0	100.0
	±	±	±	±	±	±	±	±	±	±	±	±
	0.0	0.73	0.0	0.51	0.0	0.17	0.0	0.24	0.0	0.0	0.0	0.0
Allocation time, s	4.89	86.48	6.18	81.08	9.59	163.96	6.01	102.86	16.14	258.61	19.28	221.7
	±	±	±	±	±	±	±	±	±	±	±	±
	0.07	1.21	0.07	1.5	0.06	3.77	0.05	1.34	0.14	2.65	0.16	2.58
Fragmentation, %	19.08	61.4	18.67	62.48	12.04	52.4	12.05	49.34	8.47	40.47	8.4	40.5
	±	±	±	±	±	±	±	±	±	±	±	±
	1.1	2.05	1.07	2.36	0.69	2.09	0.65	2.03	0.5	1.73	0.49	1.94
Revenue	1303.0	973.0	1303.0	1154.0	1303.0	1237.0	1303.0	1261.0	1303.0	1303.0	1303.0	1303.0
	±	±	±	±	±	±	±	±	±	±	±	±
	23.68	82.04	23.68	59.02	23.68	43.75	23.68	38.72	23.68	23.68	23.68	23.68
Link load, %	10.01	25.66	10.16	26.62	6.03	20.63	6.16	20.28	4.1	14.88	4.16	14.88
	±	±	±	±	±	±	±	±	±	±	±	±
	0.59	1.19	0.59	1.3	0.36	1.14	0.37	1.1	0.24	0.84	0.25	0.84
Node load, %	13.52	10.4	13.52	10.78	9.02	8.67	9.02	8.57	6.76	6.76	6.76	6.76
	±	±	±	±	±	±	±	±	±	±	±	±
	0.78	0.48	0.78	0.52	0.52	0.47	0.52	0.48	0.39	0.39	0.39	0.39

Abbreviations: BF, best fit; VI, virtual infrastructure; LRC, local resource capacity; WF, worst fit.

5.2.3 | Hierarchical VI requests

Hierarchical requests were composed to submit a set of VI topologies with cycles. As illustrated by VI 3 from Figure 1, hierarchical requests are composed of virtual resources organized in layers. In this sense, networking cycles are always present, regardless of the number of VMs and switches. Therefore, before allocating a hierarchical request, VITreeM must perform a graph to tree conversion.

VITreeM and DVINE results for hierarchical VI requests are shown in Figure 9 and Table 5. For this subset of requests, DVINE obtained a higher acceptance rate than VITreeM (Figure 9A). Notably, even increasing the datacenter scale from SC 3 to SC 4, VITreeM accepted approximately half of submitted requests, at most, while DVINE achieved 100%. A combined analysis of fragmentation, revenue, and load metrics indicates that VITreeM allocated small-sized requests even though the load of nodes was extremely low (Figure 9F).

TABLE 4 Results for VPC requests submitted to scenarios 5, 6, and 7

	SC 5				SC 6				SC 7			
	LRC		CENTER		LRC		CENTER		LRC		CENTER	
	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF
Acceptance rate, %	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	±	±	±	±	±	±	±	±	±	±	±	±
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Allocation time, s	4.79	5.71	5.65	5.09	10.58	13.95	12.06	11.97	19.99	24.23	17.86	24.77
	±	±	±	±	±	±	±	±	±	±	±	±
	0.13	0.23	0.07	0.14	0.14	0.44	0.13	0.59	0.32	0.86	0.2	0.88
Fragmentation, %	11.33	32.08	11.43	30.88	7.15	21.13	7.2	21.79	5.08	16.03	5.08	16.3
	±	±	±	±	±	±	±	±	±	±	±	±
	0.53	1.31	0.53	1.19	0.34	0.89	0.34	1.02	0.24	0.67	0.23	0.66
Revenue	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0	2113.0
	±	±	±	±	±	±	±	±	±	±	±	±
	97.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4
Link load, %	7.41	18.43	7.51	19.91	4.43	13.24	4.47	13.46	3.04	9.44	3.1	9.78
	±	±	±	±	±	±	±	±	±	±	±	±
	0.35	0.81	0.34	0.87	0.21	0.59	0.21	0.65	0.14	0.44	0.14	0.43
Node load, %	10.18	10.18	10.18	10.18	6.79	6.79	6.79	6.79	5.09	5.09	5.09	5.09
	±	±	±	±	±	±	±	±	±	±	±	±
	0.48	0.48	0.48	0.48	0.32	0.32	0.32	0.32	0.24	0.24	0.24	0.24

Abbreviations: BF, best fit; LRC, local resource capacity; VPC, virtual private cloud; WF, worst fit.

TABLE 5 Results for hierarchical VI requests submitted to scenarios 5, 6, and 7

	SC 5				SC 6				SC 7			
	LRC		CENTER		LRC		CENTER		LRC		CENTER	
	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF	BF	WF
Acceptance rate, %	52.5	50.32	52.5	50.29	52.5	51.68	52.5	51.68	52.5	51.68	52.5	51.68
	±	±	±	±	±	±	±	±	±	±	±	±
	1.1	1.18	1.1	1.19	1.1	1.12	1.1	1.12	1.1	1.12	1.1	1.12
Allocation time, s	9.17	23.46	106.03	27.45	28.71	59.89	437.02	68.96	90.95	166.08	1087.91	172.02
	±	±	±	±	±	±	±	±	±	±	±	±
	0.39	0.93	9.46	1.23	1.35	2.93	39.94	3.21	4.91	9.79	93.03	9.45
Fragmentation, %	15.4	30.27	15.79	29.08	10.71	23.47	10.87	23.79	8.22	17.98	7.56	18.7
	±	±	±	±	±	±	±	±	±	±	±	±
	0.65	1.05	0.68	0.96	0.5	0.97	0.48	1.02	0.38	0.77	0.36	0.79
Revenue	122.0	110.0	122.0	107.0	122.0	115.0	122.0	115.0	122.0	115.0	122.0	115.0
	±	±	±	±	±	±	±	±	±	±	±	±
	20.69	19.5	20.69	20.17	20.69	20.35	20.69	20.35	20.69	20.35	20.69	20.35
Link load, %	14.06	20.49	13.77	23.21	8.45	14.89	9.08	16.19	6.11	10.74	5.66	12.22
	±	±	±	±	±	±	±	±	±	±	±	±
	0.75	0.84	0.75	0.96	0.43	0.7	0.48	0.8	0.34	0.52	0.31	0.58
Node load, %	1.0	0.96	1.0	0.91	0.67	0.66	0.67	0.66	0.5	0.49	0.5	0.49
	±	±	±	±	±	±	±	±	±	±	±	±
	0.07	0.07	0.07	0.07	0.05	0.05	0.05	0.05	0.03	0.04	0.03	0.04

Abbreviations: BF, best fit; LRC, local resource capacity; WF, worst fit.

Moreover, an analysis focusing on physical links indicate that VITreeM was unable to use the residual capacity (Figure 9E). In other words, the conversion of large-scale VI requests to trees resulted in weighted virtual links. By moving capacity requirements to guarantee virtual resources connectivity with quality of service requirements, VITreeM created virtual links that extrapolated the residual capacity of some scenarios. Moreover, as VITreeM performed a deep search on physical tree before rejecting a request, the resulting average allocation time for large-scale scenarios (SC 5 to SC 7) pointed out the worst case of asymptotic complexity (as discussed in Section 4.4). Finally, even for large-scale

scenarios, VITreeM was unable to allocate all submitted requests (Table 5) consistently, showing high link usage regardless of the candidates ordering and starting node configuration.

5.2.4 | Discussion and key observations

Coordinated fragmentation and load decreasing leads to higher acceptance rate. A combined analysis of Figures 7 and 8 and Tables 3 and 4 depicts that the coordinated decreasing of fragmentation and load performed by VITreeM leads to higher acceptance rate and revenue. Moreover, VITreeM guided by the best-fit ordering outperformed DVINE and VITreeM with the worst-fit configuration having the highest acceptance rate. The use of the best-fit approach for sorting candidates increases the acceptance rate regardless of the approach used to define the starting node, LRC, or center.

Finally, considering the whole picture, network datacenter fragmentation induced by VITreeM with LRC and the best-fit remains nearly to values observed in previous work.[15] These results are justified because of the nature of the algorithm, which performs the allocation guided by a search in depth in the tree. The spread of virtual resources usually occurs only near the leaves. Moreover, VITreeM attempts virtual link allocation over short physical paths, placing VMs and switches as close as possible.

Tree-based search speedups VI allocation. VITreeM introduced an approach to reduce the number of comparisons performed on allocation. By grouping data on dynamically composed boxes, VITreeM avoids deep searching on subtrees. Figures 7B and 8B and Tables 3 and 4 show that VITreeM allocates multitiered and VPC requests on acceptable waiting time. In both scenarios, the number of physical candidates slightly affected allocation time. Indeed, for most scenarios, VITreeM achieves a stable performance with allocation time average of less than 5 seconds. Such performance combined with acceptance rate results indicates the possibility of using VITreeM in IaaS cloud frameworks, as requesting tenants would wait a few seconds on average.

Drawbacks of tree-based allocation. From Figure 9 and Table 5, it is evident that the graph to tree conversion performed by VITreeM decreased the acceptance rate. Albeit increasing the number of physical candidates, graph to tree conversion limits the solution space due to virtual link grouping to remove cycles.

Perspectives. VITreeM outperformed the algorithm used as base for comparison in multitiered and VPC VI requests. However, VITreeM results for hierarchical VI requests have not followed this line. Thus, as perspective for future work, the application of path splitting[11] on VITreeM can potentially increase the acceptance rate, as virtual links can be manipulated and decomposed in parallel with the boxes comparison. Considering the speedup of allocation process, VITreeM is a prominent candidate for parallelization. Finally, the definition of a starting point taking into account a global view of physical resources instead of a local one is a promising approach.[38] As subtrees information is already abstracted and represented in capacity boxes, a global resource capacity identification can analyze the entire topology, avoiding the introduction of possible bottlenecks or the creation of virtual links atop long physical paths.

6 | CONCLUSION

Cloud computing is a concept strictly present in services that are available in the internet. Among the existing services, the on-demand delivery of VIs (computing and communication resources) enables the composition of isolated and time-limited virtual entities, in which users can execute their applications. Considering management tasks performed by providers, the allocation of physical resources to host VIs is a problem belonging to NP-hard class. In this context, this paper presented VITreeM, a tree-based algorithm to allocate VIs. VITreeM converts graphs (virtual and physical) to trees and performs the allocation restricting search space by grouping information. In summary, the main contributions of this work are 4-fold: (1) a proposal to convert VIs and network datacenter graphs in trees; (2) an algorithm to reduce the search space by grouping subtrees information; (3) an online tree-based algorithm to allocate VIs; and (4) experimental results with hierarchical, multitiered, and VPC topologies. Experimental analysis indicated a promising use of VITreeM in different scenarios as it maintained a low physical infrastructure fragmentation and load combined with a high acceptance rate.

ACKNOWLEDGMENTS

The authors would like to thank LabP2D (http://labp2d.joinville.udesc.br) and UDESC for providing the resources used to implement and analyze VITreeM.

REFERENCES

1. Alshaer H. An overview of network virtualization and cloud network as a service. *Int J Network Manage*. 2015;25(1):1–30. http://dx.doi.org/10.1002/nem.1882.
2. Anhalt F, Koslovski G, Primet PV-B. Specifying and provisioning virtual infrastructures with HIPerNET. *Int J Network Manage*. 2010;20(3):129–148.
3. Rosenberg J, Mateos A. *The Cloud at Your Service*. 1st ed. Greenwich, CT, USA: Manning Publications Co.; 2010.
4. Measuring EC2 system performance. Available at http://goo.gl/V5zhEd.
5. Persico V, Marchetta P, Botta A, Pescapè A. Measuring network throughput in the cloud: the case of Amazon EC2. *Comput Networks*. 2015;93, Part 3:408–422. Cloud Networking and Communications.
6. Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl*. 2010;1(1):7–18.
7. Mell P, Grance T. The NIST definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology*, Gaithersburg, USA; 2011:1–7.
8. Fischer A, Botero J, Till Beck M, de Meer H, Hesselbach X. Virtual network embedding: a survey. *IEEE Commun Surv Tutorials*. 2013;15, Fourth(4):1888–1906.
9. Zhan Z-H, Liu X-F, Gong Y-J, Zhang J, Chung HS-H, Li Y. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput Surv*. July 2015;47(4):63:1–63:33.

10. Chowdhury N, Boutaba R. A survey of network virtualization. *Comput Networks*. 2010;54(5):862–876.
11. Yu M, Yi Y, Rexford J, Chiang M. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Comput Commun Rev*. 2008;38(2):17–29.
12. Chowdhury N, Rahman M, Boutaba R. Virtual network embedding with coordinated node and link mapping. *INFOCOM 2009, IEEE*, Rio de Janeiro, Brazil; April 2009:783–791.
13. Cheng X, Su S, Zhang Z, Wang H, Yang F, Luo Y, Wang J. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput Commun Rev*. April 2011;41(2):38–47.
14. Wang G, Zhao Z, Lu Z, Tong Y, Wen X. A virtual network embedding algorithm based on mapping tree. *13th Symp Comm and Information Technologies (ISCIT)*, Samui Island, Thailand; September 2013:243–247.
15. De S Cavalcanti G, Obelheiro R, Koslovski G. Optimal resource allocation for survivable virtual infrastructures. *10th International Conference on the Design of Reliable Communication Networks*; Ghent, Belgium; 2014:1–8.
16. Koslovski GP, Primet PV-B, Charão AS. *VXDL: virtual resources and interconnection networks description language, 138–154*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009.
17. Wolke A, Ziegler L. Evaluating dynamic resource allocation strategies in virtualized data centers. *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, Alaska, USA; June 2014:328–335.
18. Dias de Assunção M, Buyya R. Performance analysis of allocation policies for interGrid resource provisioning. *Inf Softw Technol*. 2009;51(1):42–55.
19. Zaman S, Grosu D. An online mechanism for dynamic VM provisioning and allocation in clouds. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, Honolulu, Hawaii, USA; June 2012:253–260.
20. Caggiani Luizelli M, Richter Bays L, Salete Buriol L, Pilla Barcellos M, Gaspary LP. Characterizing the impact of network substrate topologies on virtual network embedding. *Network and Service Management (CNSM), 9th Int. Conference on*, Zurich, Switzerland; August 2013:42–50.
21. Luizelli MC, Bays LR, Buriol LS, Barcellos MP, Gaspary LP. How physical network topologies affect virtual network embedding quality: a characterization study based on {ISP} and datacenter networks. *J Network Comput Appl*. 2016;70:1–16.
22. Cisco Data Center Infrastructure: 2.5 Design Guide. Cisco; 2007.
23. Stallings W. *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. 1st ed.: Addison-Wesley Professional; 2015.
24. Jennings B, Stadler R. Resource management in clouds: survey and research challenges. *J Network Syst Manage*. 2014;23:1–53.
25. de Oliveira R, Koslovski G. VITreeM - um algoritmo baseado em árvores para alocação de infraestruturas virtuais. *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas distribuídos (SBRC)*; Vitória, ES, Brazil; 2015:557–570.
26. Kreutz D, Ramos FMV, Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S. Software-defined networking: a comprehensive survey. *Proc IEEE*. January 2015;103(1):14–76.
27. Mijumbi R, Serrat J, Gorricho J-L, Bouten N, De Turck F, Boutaba R. Network function virtualization: state-of-the-art and research challenges. *IEEE Commun Surv Tutorials*. First quarter 2016;18(1):236–262.
28. Lischka J, Karl H. A virtual network mapping algorithm based on subgraph isomorphism detection. *Proc of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, ACM; Barcelona, Spain; 2009:81–88.
29. Koslovski G, Soudan S, Goncalves P, Vicat-Blanc P. Locating virtual infrastructures: users and InP perspectives. *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*; Dublin, Ireland; 2011: 153–160.
30. Sherwood R, Gibb G, kiong Yap K, Casado M, Mckeown N, Parulkar G. FlowVisor: a network virtualization layer, Technical Report; 2009.
31. Cordella LP, Foggia P, Sansone C, Vento M. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans Pattern Anal Mach Intell*. October 2004;26(10):1367–1372.
32. Mashayekhy L, Nejad M, Grosu D, Vasilakos A. Incentive-compatible online mechanisms for resource provisioning and allocation in clouds. *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, Alaska, USA; June 2014:312–319.
33. Kantarci B, Foschini L, Corradi A, Mouftah HT. Design of energy-efficient cloud systems via network and resource virtualization. *Int J Network Manage*. 2015;25(2):75–94.
34. Samuel F, Chowdhury M, Boutaba R. PolyViNE: policy-based virtual network embedding across multiple domains. *J Internet Serv Appl*. 2013;4(1):1–23.
35. Yeow W-L, Westphal C, Kozat U. Designing and embedding reliable virtual infrastructures,; 2010:33–40.
36. Chowdhury M, Rahman M, Boutaba R. ViNEYard: virtual network embedding algorithms with coordinated node and link mapping. *Networking, IEEE/ACM Trans on*. February 2012;20(1):206–219.
37. Butt NF, Chowdhury M, Boutaba R. Topology-awareness and reoptimization mechanism for virtual network embedding. *Proceedings of the 9th IFIP Networking*, IFIP; Chennai, India; 2010:27–39.
38. Gong L, Wen Y, Zhu Z, Lee T. Toward profit-seeking virtual network embedding algorithm via global resource capacity. *Proc INFOCOM*, IEEE; Toronto, Canada; 2014:1–9.
39. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. *SIGCOMM Comput Commun Rev*. August 2008;38(4):63–74.
40. West DB. *et al. Introduction to Graph Theory*, vol. 2: Prentice hall Upper Saddle River; 2001.
41. Knuth DE. The Art of Computer Programming: Fundamental Algorithms, vol. I: Addison-Wesley; 1968.
42. Ricci R, Alfeld C, Lepreau J. A solver for the network testbed mapping problem. *SIGCOMM Comput Commun Rev*. April 2003;33(2): 65–81.
43. Oliveira RR, Bays LR, Marcon DS, Neves MC, Buriol LS, Gaspary LP, Barcellos MP. DoS-resilient virtual networks through multipath embedding and opportunistic recovery. *Proc ACM SAC*, Coimbra, Portugal; 2013:597–602.

How to cite this article: de Oliveira, R., and Koslovski, G. P. (2016), A Tree-based Algorithm for Virtual Infrastructure Allocation with Joint Virtual Machine and Network Requirements *Int J Network Mgmt*., doi:10.1002/nem.1958.

AUTHORS BIOGRAPHIES

Ramon de Oliveira obtained his bachelor's degree in Computer Science at Santa Catarina State University—UDESC. Currently, he is a graduate student at the University of Campinas (UNICAMP), and his main research interests include optimization, graph theory, and machine learning.

Guilherme Piegas Koslovski is adjunct professor at Santa Catarina State University—UDESC—and holds a PhD in Computer Science from the École Normale Supérieure de Lyon (France). He obtained his graduate and master (2008) degrees in Computer Science at UFSM (Brazil). His research interests include cloud computing, elastic provisioning, cloud networking, virtual infrastructures, network functions virtualization, and software-defined networking.