A Tabu search approach for a virtual networks splitting strategy across multiple cloud providers

Marieme Diallo*, Alejandro Quintero and Samuel Pierre

Department of Computing and Software Engineering,
École Polytechnique de Montréal,
C.P. 6079, succ. Centre-Ville,
Montréal, QC,
H3C 3A7, Canada
Email: marieme.diallo@polymtl.ca
Email: alejandro.quintero@polymtl.ca
Email: samuel.pierre@polymtl.ca
*Corresponding author

Abstract: This paper addresses the problem of computational and networking resources embedding across multiple independent cloud providers (CPs). We focus on the splitting phase problem by proposing a virtual network requests (VNRs) splitting strategy, which aims at improving the performance and the quality of service (QoS) of resulting mapped VNR segments. We formalise our splitting strategy as a mathematical maximisation problem with constraints by using an integer linear program (ILP). Since the VNRs splitting process is classified as an NP-hard problem, we propose a metaheuristic approach based on the Tabu search (TS), in order to find good feasible solutions in polynomial solving time. The simulations results obtained show the efficiency of the proposed algorithm, in comparison with the exact method and an other baseline approach. Solution costs are on average close to the upper bounds, with an average gap ranging from 0% to a maximum of 2.97%, performed in a highly reduced computing time.

Keywords: cloud computing; virtualised network infrastructures; resource splitting; optimisation; metaheuristics; Tabu search.

Reference to this paper should be made as follows: Diallo, M., Quintero, A. and Pierre, S. (2020) ‘A Tabu search approach for a virtual networks splitting strategy across multiple cloud providers’, *Int. J. Metaheuristics*, Vol. 7, No. 3, pp.197–238.

Biographical notes: Marieme Diallo received her Bachelor’s in Computer Engineering from the Université Gaston Berger de Saint-Louis, Senegal, in 2008, and Master’s in Computer Engineering from the École Polytechnique de Montréal, Canada, in 2014. She has been a member of the Mobile Computing and Networking Research Laboratory at École Polytechnique de Montréal, Canada, since May 2010, where she is currently working towards her PhD in Computer Engineering. Her research interests include cloud computing, network virtualisation and optimisation.

Alejandro Quintero received his Engineer's in Computer Engineering from the Los Andes, Colombia, in 1983. In June 1989 and in 1993, he received his diploma of Advanced Studies and Doctorate in Computer Engineering, respectively, from the INPG Grenoble and Université Joseph Fourier, Grenoble, France. He is currently a Full Professor at the Department of Computer Engineering of École Polytechnique de Montréal, Canada. His main research interests include services and applications related to mobile computing, network security, networks infrastructures and next generation mobile networks. He is the co-author of one book, as well as more than 60 other technical publications including journal and proceedings papers.

Samuel Pierre is currently a Professor in the Department of Computer and Software Engineering at the École Polytechnique de Montréal, and the Director of the Mobile Computing and Networking Research Laboratory (LARIM). His research interests include wired and wireless communications, mobile computing and networking, cloud computing and e-learning. He received several awards, including the Prix Poly 1873 for excellence in teaching and training (2001 and 2005), and the Knight of the National Order of Quebec in 2009. In December 2011, he was appointed as a member of the Order of Canada. In May 2014, he received his Honorary Doctorate from the Université du Quebec à Trois-Rivieres (UQTR) and another is from the Université du Quebec en Outaouais (UQO) in November 2016.

1 Introduction

Cloud computing has recently emerged as an innovative utility computing solution (Armbrust et al., 2010), allowing small businesses to rent distributed configurable resources as an on-demand service model. Nowadays, the Infrastructure as a Service (IaaS) has become the most widely adopted cloud service model (Manvi and Shyam, 2014). In this paradigm, a service provider (SP) can lease a large pool of virtualised infrastructure layer resources (computational and networking) from one or more cloud providers (CPs), in order to build heterogeneous virtual networks that will offer customised services to its end clients.

In the IaaS business model, a fundamental management problem lies in the efficient embedding of co-existing virtual network requests (VNRs) onto distributed substrate infrastructures (Belbekkouche et al., 2012; Manvi and Shyam, 2014). This issue, usually referred to as the well-known NP-hard virtual network embedding (VNE) problem (Chowdhury et al., 2012; Fischer et al., 2013; Zhang et al., 2016), becomes more challenging when the substrate infrastructures are owned by multiple independent CPs (Grozev and Buyya, 2012; Rafael et al., 2012). Indeed, in such a context, the VNE process requires two major phases of operation, each of them dealing with an NP-Hard problem with different technical approaches to resolve it: the multi-cloud VNRs splitting phase, followed by the intra-cloud VNR segments mapping phase. In the first phase, which is similar to a graph partitioning problem (Sanchis, 1989; Tao et al., 1992), a virtual network provider (VNP), acting as a virtual brokerage service on behalf of the SP (Fischer et al., 2013), generally uses a strategy to select eligible CPs based on the SPs requirements, and split the VNRs into different segments. In the second phase, which corresponds to the VNE problem within a single CP (Chowdhury et al., 2012; Khan et al., 2016; Zhang et al., 2014), each selected CP uses a mapping approach to embed the assigned VNR segments into its intra-cloud network.

VNE over a multi-cloud network has been only recently addressed in the literature (Dietrich et al., 2015; Leivadeas et al., 2013; Mano et al., 2016; Mechtri et al., 2015). It adds more complexity and scalability issues. Indeed, due to the non-interoperability between CPs, it becomes difficult to have optimal configurations from the global view of the multi-cloud embedding process. Moreover, the VNP visibility on the multi-cloud substrate network is essentially decisive for the efficiency of any VNRs splitting strategy. However, the VNP proceeds with the splitting phase based on a very poor knowledge of the multi-cloud environment. Information such as substrate network topologies and details about the resources availabilities are usually concealed by the CPs (Dietrich et al., 2015; Mano et al., 2016). The VNP is then restricted to CP policies, while generating embedding solutions which must best satisfy the SPs requirements. On the other hand, the few early heuristics-based solutions on the multi-cloud VNE problem have generally assumed that CPs would disclose some private information (Leivadeas et al., 2013; Samuel et al., 2013). Some others aim only at minimising the resource provisioning price for the SP during the splitting phase (Dietrich et al., 2015; Mano et al., 2016), regardless of the performance and quality of service (QoS) of resulting embedded VNR segments.

In this paper, we propose a QoS-based splitting strategy for a VNRs embedding across multiple IaaS providers. Our approach considers resource and QoS constraints based on SPs requirements, with the purpose of splitting efficiently the VNRs and improving the performance of resulting VNR segments that are mapped onto the selected intra-cloud infrastructures. The key contributions of this work are as follows:

- We have designed a multi-cloud VNRs splitting approach based on the limited information disclosed by the CPs and the dynamic and heterogeneous nature of the substrate network. In addition, for a better efficiency of our splitting strategy, we have enriched the visibility of the VNP on the multi-cloud environment, by taking advantage of certain information related to network topologies that is not treated as confidential, such as CPs point-of-presence (PoP) (Dietrich et al., 2015).
- We use an integer linear programming (ILP) model to formalise the proposed strategy as a maximisation problem with constraints. Resource provisioning costs are defined based on the availability of supplied resources and the performance guarantees advertised by the CPs.
- Taking into account the NP-hard nature of the VNRs splitting phase problem, we propose a Tabu search (TS) algorithm (Glover, 1989, 1990), in order to solve large instances of the problem in polynomial computing time.
- To handle the VNR segments mapping phase, we have adopted a multi-objective intra-cloud mapping approach for each selected CP, in order to minimise mainly the overall delay. The adopted approach follows the work of Larumbe and Sanso (2013), which formalises the mapping problem as a mixed-integer linear programming (MILP).

The rest of the paper is organised as follows: Section 2 discusses relevant related work. Section 3 describes the multi-cloud VNE problem and the proposed embedding framework. Section 4 presents the mathematical formulation related to the VNRs splitting problem. Section 5 presents the adaptation of the proposed algorithm. Performance evaluation and simulation results are presented in Section 6. Finally, conclusion and future works are highlighted in Section 7.

2 Related work

VNE represents the main resource allocation challenge in cloud infrastructures virtualisation (Belbekkouche et al., 2012; Manvi and Shyam, 2014). It is known to be an NP-hard problem (Chowdhury et al., 2012; Zhang et al., 2016). The problem consists on a simultaneous and optimised mapping with constraints of virtual nodes resources and virtual links (VLs) resources onto the substrate networks, which is generally reduced to the NP-hard multiway separator problem (Fischer et al., 2013). As a result, various heuristic-based algorithms have been proposed, in order to satisfy economic benefits, resource utilisation-efficiency, energy-efficiency, survivability and QoS aspects. Indeed, exact approaches are not scalable and can generally only be applicable in small sized scenarios (Dietrich et al., 2015; Houidi et al., 2011, 2015; Mechtri et al., 2015).

Here we discuss related work on VNE over a single-cloud network and VNE over a multi-cloud network.

2.1 VNE over a single-cloud network

Many algorithms, mostly based on (meta) heuristic approaches, have been proposed for the single-domain VNE optimisation problem. Some works tend to solve the problem by providing a certain coordination between the node mapping stage and the link mapping stage (Chowdhury et al., 2012). Hesselbach et al. (2016) recently proposed a new path algebra-based embedding strategy that coordinates in a single step the mapping of nodes and links.

Dynamic methods, which support remappings of resources during the life-time of requests, have also been suggested. Ayoubi et al. (2015) proposed a Tabu-based framework for reliable VNE with migration, which consists of availability-aware resource allocation and reconfiguration. Rahman and Boutaba (2013) proposed a fast re-embeddings strategy based on a hybrid policy heuristic that aims at network survivability. Zhang et al. (2014) took a step further by introducing a first-fit-based approach to support opportunistic resource sharing among virtual nodes and VLs. Because most of these approaches may suffer from improper load balancing and link under-utilisation, Khan et al. (2016) proposed a proactive and reactive multi-path link embedding approach to achieve the VNE survivability, while minimising both resource redundancy and path splitting overhead.

Other works introduce multi-objective approaches to resolve the problem. Larumbe and Sanso (2012, 2013) proposed an efficient multi-objective model for mapping virtual resources into various locations of cloud data centres. Their solution is based on a TS approach that allows CPs to minimise delay, environmental and resource operating costs. In the same way, Melo et al. (2013) proposed an ILP model to solve the online VNE problem in order to minimise the resource consumption, while performing load balancing. Their work was extended in Melo et al. (2015), by considering in addition the minimisation of energy consumption. Houidi et al. (2015) also proposed a multi-objective fault-tolerant VNE algorithm formalised as a MILP, which takes into account constraints related to power consumption, resource availability and load balancing.

These intra-domain VNE solutions, although they may be optimal, can not be properly applicable to a multi-domain VNE problem. They are performed by the CP based on a complete knowledge of its substrate network topologies and available resource distributions. As a result, upon splitting the VNRs, they can only be used by each selected CP for the VNR segments intra-cloud mapping phase.

2.2 VNE over a multi-cloud network

VNE over a multi-domain has not been intensively studied. Research works have essentially focused on the VNRs splitting phase, since existing solutions related to the single-domain VNE problem are usually adopted for the mapping phase.

Houidi et al. (2011) were one of the first authors working in this research field. They combined an exact ILP method with a heuristic algorithm based on recursive max-flow min-cut to find the optimal VNRs splitting across multiple CPs. Samuel et al. (2013) introduced a distributed protocol for VNE problem in multiple substrate networks. Their approach aims at coordinating the participating CPs through competitive pricing mechanisms, in order to maximise their revenue. The proposal of Leivadeas et al. (2013) is one of the few performance and efficiency-based approaches for VNRs splitting. The authors proposed an hierarchical framework where the VNRs splitting problem is solved through an iterated local search (ILS) algorithm. However, even though costs are not randomly generated as with Houidi et al. (2011), their approach does not respect certain information concealed by the CPs (Dietrich et al., 2015). Also, their resource provisioning costs definition lacks on the heterogeneous nature of the substrate network. Mechtri et al. (2015) proposed a heuristic algorithm based on graph decomposition into topology patterns and bipartite graph matching, in order to solve the resources mapping problem in distributed and hybrid cloud environments.

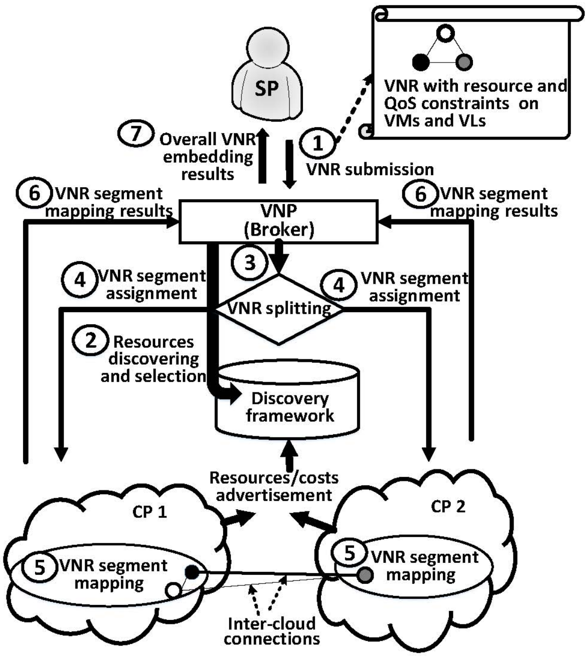
On the other hand, some works have considered the limited level of access to information of the VNP before proceeding to the splitting phase. Dietrich et al. (2015) have studied the suboptimality of multi-domain VNE with limited information disclosure, whose the efficiency is compared with the 'best-case' scenario where the complete network topology and resource availability information is accessible by the VNP. Mano et al. (2016) proposed also a novel optimisation method restricted to CPs private information, which uses a secure multi-party computation (MPC) to generate minimal operations that minimise inter-domain VNE prices. However, both proposals are only focus on VNRs splitting at the lowest price for the SP, without considering performance and QoS of embedded VNR segments.

Based on the literature, most VNRs splitting strategies across multiple CPs are only based on exact methods, and thus can only scale up to topologies with small sizes (Dietrich et al., 2015). Some others based on heuristics and using a more sophisticated splitting scheme (Leivadeas et al., 2013; Samuel et al., 2013) assume that CPs would share some private information with the VNP. Not to mention that most proposals, even though they are restricted to the domain-privacy of CPs (Dietrich et al., 2015; Gong et al., 2016; Li et al., 2016; Mano et al., 2016), tend only to minimise the resource provisioning price for the SP. This may not give opportunities to the SP to select CPs based only on desired performance and QoS.

3 Multi-cloud VNE problem

In this section, we first briefly describe the problem of VNE across multiple CPs. Then, we present an overview of the proposed framework, where an hierarchical approach is generally adopted to decompose the global problem into the VNRs splitting phase and the VNR segments mapping phase.

Figure 1 Multi-cloud VNE process


SP
VNR with resource and QoS constraints on VMs and VLs
7 Overall VNR embedding results
1 VNR submission
6 VNR segment mapping results
6 VNR segment mapping results
VNP (Broker)
3
4 VNR segment assignment
4 VNR segment assignment
VNR splitting
2 Resources discovering and selection
Discovery framework
Resources/costs advertisement
CP 1
CP 2
5 VNR segment mapping
5 VNR segment mapping
Inter-cloud connections

3.1 Problem description

The global multi-cloud VNE process is described in Figure 1. The problem consists in efficiently embedding VNRs onto multiple cloud networks owned by independent CPs. VNRs are submitted in a high level of abstraction by the SP, as a set of virtual nodes interconnected by VLs representing the exchanged traffic between virtual nodes (Fischer et al., 2013). Virtual nodes and VLs are specified with constraints related to required resources and QoS criteria (e.g., processing, memory, storage, bandwidth, QoS parameters, geographic footprint, etc.), that the embedding process has to satisfy. The SP will generally rely on the VNP, which acts as a broker to discover and select a set of advertised resources, assembled from multiple CPs and stored in a discovery framework (Lv et al., 2010). Heterogenous resources are then allocated to host virtual nodes (e.g., VMs) in specific substrate nodes (e.g., data centres), and to route VLs onto substrate paths. In most cases, to meet the SPs requirements, the VNP will need to optimally split the VNRs among eligible CPs. The resulting VNR segments are then mapped by each selected CP, and are subsequently interconnected via appropriate inter-cloud links (Leivadeas et al., 2013).

3.2 Multi-cloud VNE framework

Here, an analysis of the resource discovery framework is presented, followed by a summary description of the approaches proposed to solve each of the splitting and mapping phases.

3.2.1 Resource discovery framework and assumptions

The VNP visibility on the multi-cloud substrate network is essentially decisive at the splitting phase. An effective splitting strategy should conduct a thorough investigation into the discovery of resource information, in order to avoid inefficient embedding solutions.

A CP typically classifies its substrate nodes into different categories (e.g., Amazon, 2017), each of them having common set of functional attributes and associated with a set of computational resource types (Leivadeas et al., 2013). Node functional attributes define characteristics and properties related to the node type, operating system, virtualisation environment, geographic footprint, QoS parameters, etc. Computational resource types can be the CPU, disk space, memory, etc., each of them associated with a certain capacity that the CP keeps dynamically updated (Houidi et al., 2011). Similarly, substrate links are also classified into different types (e.g., VLAN, L3/L2 VNP, etc.) and are associated with a bandwidth capacity dynamically updated too. However, as previously mentioned, detailed information such as substrate network topologies, router-level connectivity, transit network topologies, number of data centres located in a cloud and their interconnectivity, number of instances of available resources and their utilisation, are not disclosed to the VNP. Nevertheless, some information about transit networks (PoPs) of CPs can be accessible in a high level of abstraction (Dietrich et al., 2015; Doverspike et al., 2010), allowing the VNP to expand its limited view on the substrate multi-cloud network. Those information can include an oversimplified view of PoP networks, traffic statistics and some performance guarantees for a link such as delay bounds (Larumbe and Sanso, 2013; Lv et al., 2010). A CP can also supply the maximum or the minimum capacity of a resource it can offer or guarantee (Chaisiri et al., 2012), without revealing details about the distribution and utilisation of these resources inside its intra-cloud network.

Based on these observations, in our framework the following information are considered accessible by the VNP:

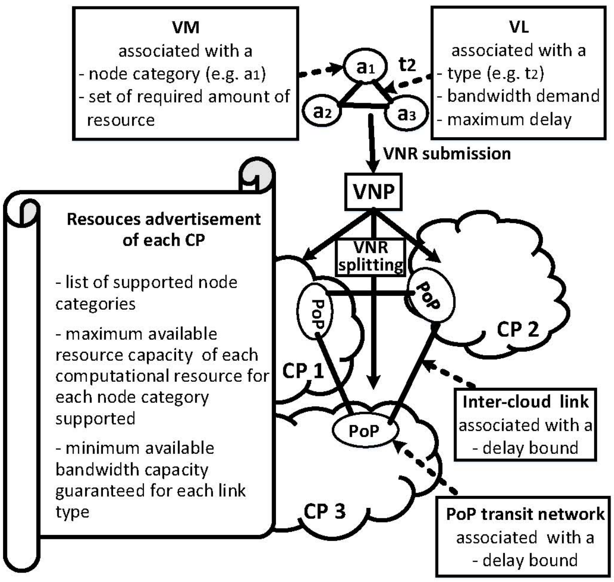
1 the maximum available capacity of each computational resource type a CP can offer for each category of node

2 the minimum available bandwidth capacity a CP can guarantee in its intra-domain for each type of link

3 the estimated delay bound on an inter-cloud link and for transiting through a PoP network.

Without loss of generality, we consider that virtual nodes are virtual machines (VMs) packaged by the SP. A PoP network is also represented in a high level of abstraction as a macro node associated with a transiting delay bound. For sake of simplicity, we only consider one PoP network per CP. CPs are interconnected in a half mesh connectivity and we did not consider in this work the path splitting scenario, as the VNP does not know the residual bandwidth capacity of inter-cloud paths. Note that CPs may also advertise the unit monetary cost of resources they offer in the discovery framework. But, in this present work, the expenditure minimisation for the SP is not our interest. We only focus on an efficient VNE that targets the performance and the QoS of split VNRs.

Figure 2 VNRs splitting phase based on the resource discovery framework


VM
associated with a
- node category (e.g. a1)
- set of required amount of resource
a1
t2
a2
a3
VL
associated with a
- type (e.g. t2)
- bandwidth demand
- maximum delay
VNR submission
VNP
VNR splitting
Resouces advertisement of each CP
- list of supported node categories
- maximum available resource capacity of each computational resource for each node category supported
- minimum available bandwidth capacity guaranteed for each link type
PoP
PoP
PoP
CP 1
CP 2
CP 3
Inter-cloud link
associated with a
- delay bound
PoP transit network
associated with a
- delay bound

3.2.2 Multi-cloud VNRs splitting phase

The multi-cloud VNRs splitting phase is illustrated in Figure 2. Upon receiving an incoming VNR, the VNP relies on the above disclosed information to identify eligible substrate resources able to fulfill the SPs requirements. To simplify this matching step, we assume that VMs and VLs are classified and specified with characteristics at the same level as substrate nodes and substrate links respectively. Furthermore, we consider that VLs are associated with a maximum communicating delay, in order to allow the QoS fulfilment for delay-sensitive applications. Resources required by a VNR may be supplied by more than one CP, at different provisioning costs. In our framework, the VNP will perform a VNRs splitting strategy, by using a TS-based approach to evaluate the most cost-effective resource provisioning depending on their availability, while considering QoS restrictions specified in the request. Thereafter, the VNP sends the resulting VNR segments to the selected CPs so they can proceed to the intra-cloud mapping phase.

TS metaheuristic is adopted in this framework mainly due its efficiency in achieving optimal or near-optimal solutions for various optimisation problems that deal with scalability issues.

3.2.3 Intra-cloud VNR segments mapping phase

Each CP receiving a particular VNR segment maps it onto its infrastructure by using an appropriate intra-cloud mapping method. In our framework, we adopt a multi-criteria solution based on the principles stated by Larumbe and Sanso (2012, 2013). In this approach, a selected CP must map in the best case all the assigned request segments, by using a multi-objective MILP model. The latter aims to minimise the total embedding cost (including resource, traffic and environmental costs), while providing the best possible QoS by reducing mainly the overall traffic routing delay. However, we have re-stated the approach in order to remain consistent with the level of granularity we have considered in this work, which is limited to data centres and not to physical servers hosted into the data centres.

4 System modelling and mathematical formulation

In this section, the notation used to describe the multi-cloud VNRs splitting problem is presented, followed by the mathematical formulation related to the proposed strategy. The modelling, notation and formulation related to the intra-cloud VNR segments mapping problem are available in Appendix.

4.1 Notation

The notation is composed of sets, parameters and variables describing the multi-cloud substrate network, the VNR, the costs definition and the decision variables:

- *Global sets:* Let A designate the set of node categories, R the set of computational resource types and T the set of link types.
- *Multi-cloud substrate network:* The substrate network is modeled as a weighted undirected graph $G^S = (N^S, L^S)$, where N^S is the set of substrate nodes and L^S the set of substrate links (inter-cloud links). Let I designate the set of CPs and Θ the set of all PoP nodes (transit networks). Note that N^S includes I and Θ, and PoP nodes are only used to access the network of a CP or to transit through it. The maximum available capacity of resource $r \in R$ that CP i can offer for nodes of category $a \in A$ is denoted by Q_{rai} ($Q_{rai} \in \mathbb{N}$). The minimum available bandwidth capacity that CP i can guarantee in its intra-cloud network for links of type $t \in T$ is denoted by B_{ti} ($B_{ti} \in \mathbb{N}$). The average delay bound of inter-cloud link $e \in L^S$ and for transiting through PoP network $p \in \Theta$ are respectively denoted by d_e and d_p. Let $\mathcal{P}_{ij}$ designate the set of all paths between CP i and CP j. We denoted by $\mathcal{O}_\varphi$ the set of all PoP transit networks in Θ spanned by path $\varphi \in \mathcal{P}_{ij}$.

- *VNR modelling:* The VNR is also modeled as a weighted undirected graph $G^V = (N^V, L^V)$, where N^V is the set of VMs and L^V the set of VLs representing the inter-VMs traffic. Let N_a^V denote the set of VMs defined in category of node $a \in A$ ($\bigcup_{a \in A} N_a^V = N^V$). The required amount of resource $r \in R$ by VM v is denoted by q_{rv} ($q_{rv} \in \mathbb{N}_1$). Let q_r^{min} and q_r^{max} designate respectively the lowest and the highest amount of resource $r \in R$ ever requested in a VNR. Let L_t^V denote the set of VLs of type $t \in T$ ($\bigcup_{t \in T} L_t^V = L^V$). The bandwidth demand and the maximum allowed traffic delay of VL l are respectively denoted by $b_l (b_l \in \mathbb{N}_1)$ and δ_l. This maximum delay adds constraints on the assignment of communicating VMs, in order to avoid delay violations on VLs.
- *Cost parameters:* We denote by $\mathcal{C}_{rai}^N$ the node provisioning cost of CP i for nodes of category $a \in A$ for resource of type $r \in R$. Let $\mathcal{C}_{ti}^L$ designate the intra-cloud link provisioning cost of CP i for links of type $t \in T$, and $\mathcal{C}_{t\varphi}^L$ the inter-cloud link provisioning cost of path $\varphi \in \mathcal{P}_{ij},\ i, j \in I$, for links of type $t \in T$.
- *Other parameters:* We defined other parameters whose equations are stated in Subsection 4.2.1. Let Q_{ra}^{max} denote the highest available capacity of resource $r \in R$ supplied among all CPs for nodes of category $a \in A$. Let B_t^{max} denote the highest available bandwidth capacity guaranteed among all CPs for links of type $t \in T$. Let w_{rv} designate the weight of resource $r \in R$ required by VM v. Let α and β denote respectively the node demand weight and the link demand weight in the total VNRs splitting cost.
- *Decision variables:* Let X_{vi} denote the binary variable set to 1 if VM v is assigned to CP i and 0 otherwise. Let $Y_{l\varphi}$ denote the binary variable set to 1 if VL l is assigned to path $\varphi \in \mathcal{P}_{ij},\ i, j \in I,\ t \in T$, and 0 otherwise.

4.2 VNRs splitting problem formulation

Our splitting strategy aims to design a VNE method with the best performance of resulting VNR segments, while satisfying resources and QoS requirements. To this end, we define the intra-cloud resource provisioning cost based on the availability of supplied resources. The provisioning cost of an inter-cloud path is stated as a function of the intra-cloud link provisioning cost of the two corresponding endpoints CPs, and the number of PoP transit networks the path spans.

4.2.1 Multi-cloud resource provisioning costs definition

We first define by equation (1) the parameter Q_{ra}^{max}. The node provisioning cost per CP is then given by equation (2), as the maximum available capacity of each resource the CP can offer for each category of nodes, normalised into the range [0 1] by dividing it by Q_{ra}^{max} (or by the value 1 to avoid division by zero):

$$Q_{ra}^{max} = \max_{i \in I} \left(Q_{rai} \right), \quad \forall\, r \in R,\ a \in A \tag{1}$$

$$C_{rai}^N = \frac{Q_{rai}}{\max\left(Q_{ra}^{max}, 1 \right)}, \ \forall\, r \in R,\ a \in A,\ i \in I \tag{2}$$

The parameter w_{rv} is given by the following, where the values of q_r^{max} and q_r^{max} are estimated and kept updated by the VNP based on previous statistics:

$$w_{rv} = \frac{q_{rv} - q_r^{min}}{q_r^{max} - q_r^{min}}, \quad \forall\, r \in R,\ v \in N^V, \quad \begin{array}{l} q_r^{min},\ q_r^{max} \in \mathbb{N}_{\mathbb{1}} \\ q_r^{min} \le q_{rv} \le q_r^{max} \end{array} \tag{3}$$

The parameter B_t^{max} is defined by equation (4). The intra-cloud link provisioning cost per CP is then given by equation (5), as the minimum available bandwidth capacity the CP can guarantee for each link type, normalised into the range [0 1] by dividing it by B_t^{max} (or by the value 1 to avoid division by zero):

$$B_t^{max} = \max_{i \in I} \left(B_{ti}\right), \quad \forall\, t \in T \tag{4}$$

$$C_{ti}^L = \frac{B_{ti}}{\max(B_t^{max}, 1)}, \quad \forall\, t \in T,\ i \in I \tag{5}$$

The inter-cloud link provisioning cost of each path between a pair of CPs is given by equation (6), as the minimum intra-cloud link provisioning cost between the two endpoint CPs of the path, divided by the number of PoP transit networks spanned:

$$C_{t\varphi}^L = \frac{\min\left(C_{ti}^L, C_{tj}^L\right)}{\max\left(\left|\mathcal{O}_\varphi\right|, 1\right)}, \quad \begin{array}{l} \forall\, t \in T,\ i, j \in I, \varphi \in \mathcal{P}_{ij}. \\ \text{with } \forall \varphi \in \mathcal{P}_{ii},\ i \in I,\ \mathcal{O}_\varphi = \emptyset \end{array} \tag{6}$$

Note that if the two endpoints of a path represent the same CP (i.e. i is equal to j), equation (6) will be equivalent to the corresponding CPs intra-cloud link provisioning cost stated in equation (5). Since our objective function is a cost maximisation, equation (6) will guide a solution to prefer the intra-cloud paths, otherwise the inter-cloud paths spanning the least possible PoP transit networks.

Since the number of VLs is usually much higher than the number of interconnected VMs, we need to balance the node cost and the link cost in our objective function. To this end, we define the parameters α and β as follows:

$$\alpha = \frac{|L^V|}{|N^V| + |L^V|}, \qquad \beta = 1 - \alpha \tag{7}$$

4.2.2 ILP formulation

The objective function of the proposed ILP model is expressed as follows:

MAX

$$\alpha \sum_{i \in I} \sum_{r \in R} \sum_{a \in A} \sum_{v \in N_a^V} w_{rv} C_{rai}^N X_{vi} + \beta \sum_{i \in I} \sum_{j \in I} \sum_{\varphi \in \mathcal{P}_{ij}} \sum_{t \in T} \sum_{l \in L_t^V} C_{t\varphi}^L Y_{l\varphi}, \tag{8}$$

The first term of the objective function represents the total node provisioning cost for assigning VMs to CPs. The second term represents the total link provisioning cost for assigning VLs to intra-cloud paths (i.e., where i is equal to j) or inter-cloud paths (i.e., where i is different from j). Note that the objective function is maximised since the evaluated cost represents a benefit for the SP.

The model is subjected to the following constraints:

$$\sum_{i \in I} X_{vi} = 1, \quad \forall\, v \in N_a^V,\ a \in A \tag{9}$$

Constraint (9) ensures that each VM must be assigned to exactly one CP.

$$Y_{l\varphi} \leq \frac{X_{ui} + X_{vj}}{2}, \forall\, l = uv \in L_t^V,\ t \in T,\ u, v \in N_a^V (v \neq u),\ a \in A, \\ \varphi \in \mathcal{P}_{ij},\ i, j \in I \tag{10}$$

Constraint (10) states the binary value of the variable $Y_{l\varphi}$ for each VL between two communicating VMs.

$$\sum_{i \in I} \sum_{j \in I} \sum_{\varphi \in \mathcal{P}_{ij}} Y_{l\varphi} = 1, \ \forall\, l \in L_t^V,\ t \in T \tag{11}$$

Constraint (11) ensures that each VL must be assigned to exactly one intra-cloud path, otherwise to an unique inter-cloud path.

$$\sum_{v \in N_a^V} q_{rv} X_{vi} \leq Q_{rai},\ \forall\, r \in R,\ a \in A,\ i \in I \tag{12}$$

Constraint (12) ensures that the total amount of a resource required by all VMs assigned to a CP, must not exceed the maximum available capacity of the resource offered.

$$\sum_{i \in I} \sum_{j \in I} \sum_{\varphi \in \mathcal{P}_{ij}} d_\varphi Y_{l\varphi} \leq \delta_l, \forall\, l \in L_t^V,\ t \in T \tag{13}$$

where

$$d_\varphi = \sum_{e \in \varphi} d_e \ + \sum_{p \in O_\varphi} d_p, \forall\, \varphi \in \mathcal{P}_{ij},\ i, j \in I \tag{14}$$

Constraint (13) ensures that the total delay on a path to which a VL is assigned must be less than the maximum delay allowed for the VL.

$$X_{vi} \in \{0, 1\} \quad \forall\, v \in N_a^V,\ a \in A,\ i \in I \tag{15}$$

$$Y_{l\varphi} \in \{0, 1\} \quad \forall\, l \in L_t^V,\ t \in T,\ \varphi \in \mathcal{P}_{ij},\ i, j \in I \tag{16}$$

Constraints (15) and (16) express the binary domain of each variable.

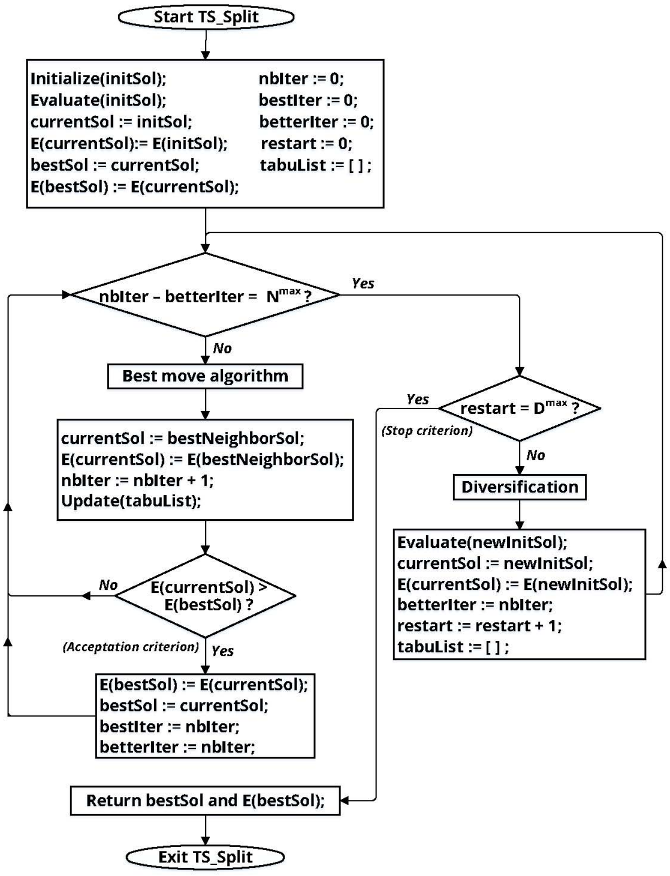
5 The proposed TS splitting algorithm

In this section, we present the basic principles of the TS algorithm, followed by the proposed adaptation of the metaheuristic, named *TS_Split*, in order to efficiently solve the VNRs splitting problem for large sized instances.

5.1 Basic principles of TS

The TS algorithm uses a local search (LS) method which explores the solution space by moving iteratively from a solution S to the best solution in the neighbourhood of S. Contrary to classical descent methods, to overcome local optima and cycling problems, the TS method allows moves that deteriorate the actual best solution and stores temporarily forbidden moves in a Tabu list. However, a Tabu move can occasionally be allowed if it satisfies a specific aspiration criterion. The search stops whenever a given stop criterion is satisfied. The TS method can also be enhanced by using the adaptive memory mechanisms.

Figure 3 The proposed TS algorithm



5.2 TS_Split algorithm

The proposed algorithm is presented in Figure 3. It combines a LS process and a long-term memory mechanism. *TS_Split* starts with an initial solution $initSol$. Then, at each iteration, the LS process tries to improve the best solution $bestSol$, by only accepting current improving solutions $currentSol$. If $bestSol$ has not been improved after N^{max} iterations, the long-term memory mechanism is then performed. The latter generates a new initial solution $newInitSol$, from which the LS process restarts the search in order to find a better solution. *TS_Split* algorithm stops after D^{max} runs of the *diversification* mechanism. The parameters of the algorithm are defined in Subsection 6.2 as results of parameterisation tests. The solution space, the initial solution, the LS process, as well as the long-term memory mechanism are detailed in the following.

5.2.1 Solution space

A solution S of *TS_Split* is an assignment of each VM of a VNR to a unique CP, and the assignment of each VL to a unique intra-cloud or inter-cloud path. Thus, a solution is determined by the setting of variables X_{vi} and $Y_{l\varphi}$ that satisfies the constraints of the problem, particularly the ones stated in equation (12) and equation (13). To avoid unfeasible solutions, two negative penalty costs $P^N(S)$ and $P^L(S)$, related respectively to the possible violation of constraints (12) and (13), are added to the objective function. Thereby, the cost of a solution S is defined by the evaluation cost $E(S)$, which includes the intrinsic cost $C(S)$ related to the objective function in equation (8), plus the penalty costs $P^N(S)$ and $P^L(S)$. At each iteration, the LS process accepts better solutions S that improves $E(S)$.

5.2.2 Initial solution

In order to generate the initial solution $initSol$, three methods were tested:

1 the highest node provisioning cost method, where VMs are sorted in descending order of their average weight of all requested resources and then assigned to the first suitable CP that has the highest node provisioning cost

2 the single-CP assignment method, where all VMs are assigned to a randomly selected CP

3 the random generating of initial solution, where each VM is randomly assigned to a CP and each VL is assigned to the shortest path between the corresponding pair of CPs.

Note that any of these methods can guaranty a feasible $initSol$ simultaneously for VMs and VLs. Best solutions were noticed with the third method. Therefore, a totally random initial solution were considered in our algorithm.

5.2.3 LS process

Each iteration of the LS process consists of moving from a current solution $currentSol$ to its best neighbour solution, by choosing the best move to apply to the current solution. In the following, we describe our best move algorithm, which defines the best neighbour $bestNeighbourSol$ of $currentSol$, and how the list of temporarily Tabu (forbidden) moves $TabuList$ is updated after selecting the best move.

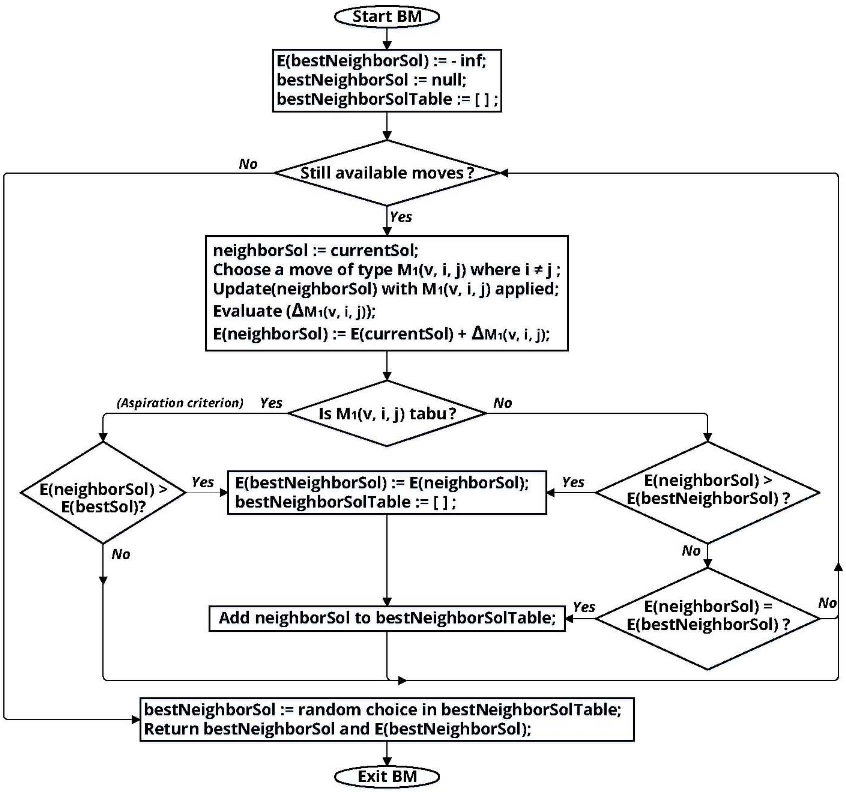
Best move algorithm

Two different types of moves were examined in this work:

- $M_1(v, i, j)$ which moves VM v from its actual assigned CP i to a different CP j
- $M_2(v_1, v_2)$, which swaps VM v_1 and VM v_2, respectively assigned to different CPs

The swapping move $M_2(v_1, v_2)$ was computationally more expensive than the simple move $M_1(v, i, j)$. With the established parameters of our algorithm, the simple move was globally as efficient as the swapping move in finding near-optimal solutions with a much shorter calculation time. For this reason, only the simple move was applied in our algorithm.

Figure 4 The best move algorithm


Start BM
E(bestNeighborSol) := - inf;
bestNeighborSol := null;
bestNeighborSolTable := [ ];
Still available moves?
No
Yes
neighborSol := currentSol;
Choose a move of type M1(v, i, j) where i ≠ j ;
Update(neighborSol) with M1(v, i, j) applied;
Evaluate (ΔM1(v, i, j));
E(neighborSol) := E(currentSol) + ΔM1(v, i, j);
(Aspiration criterion) Yes
Is M1(v, i, j) tabu?
No
E(neighborSol) > E(bestSol)?
Yes
E(bestNeighborSol) := E(neighborSol);
bestNeighborSolTable := [ ];
Yes
E(neighborSol) > E(bestNeighborSol) ?
No
No
Add neighborSol to bestNeighborSolTable;
Yes
E(neighborSol) = E(bestNeighborSol) ?
No
bestNeighborSol := random choice in bestNeighborSolTable;
Return bestNeighborSol and E(bestNeighborSol);
Exit BM

The best move algorithm is presented in Figure 4. It explores the entire neighbourhood of $currentSol$ while there is still available moves. The move $M_1(v, i, j)$ that generates the highest evaluation cost $E(bestNeighbourSol)$, depending on its Tabu status and the aspiration criterion, is chosen. The aspiration criterion allows a Tabu move if only the latter improves $bestSol$. Also, instead of selecting the first (or the last) best move after evaluating all the available moves, our algorithm considers all the moves that generate the same highest $E(bestNeighbourSol)$ and randomly selects one of them.

Furthermore, instead of evaluating the entire objective function and the penalty costs for each neighbour solution, only the evaluation cost difference $\Delta_{M_1(v,i,j)}$ generated by the move from the current solution to its neighbour solution is calculated. This approach is more complicated to implement but it decreases significantly the computing time, by reducing the complexity order of cost calculation from $O(n^2)$ to $O(n)$, with n representing the number of VMs in a request.

Let $neighbourSol$ denote a neighbour solution of $currentSol$ after applying a move $M_1(v,i,j)$ on $currentSol$. The gain $\Delta_{M_1(v,i,j)}$, with $v \in N_a^V,\ a \in A,\ i,j \in I,\ j \neq i,$ is defined as follows:

$$\Delta_{M_1(v,i,j)} = \alpha\left(\Delta^N_{M_1(v,i,j)} + \Delta^{P^N}_{M_1(v,i,j)}\right) + \beta\left(\Delta^L_{M_1(v,i,j)} + \Delta^{P^L}_{M_1(v,i,j)}\right) \tag{17}$$

$\Delta^N_{M_1(v,i,j)}$ represents the intrinsic node cost difference from $currentSol$ to $neighbourSol$, given by:

$$\Delta^N_{M_1(v,i,j)} = \sum_{r\in R} w_{rv}\left(C^N_{raj} - C^N_{rai}\right) \tag{18}$$

$\Delta^{P^N}_{M_1(v,i,j)}$ represents the node penalty cost difference from $currentSol$ to $neighbourSol$, given by:

$$\Delta^{P^N}_{M_1(v,i,j)} = -10^5 \times \sum_{r\in R}\left(\begin{array}{l} \max\left(0, \displaystyle\sum_{\substack{w \in N_a^V \\ w \neq v}} (q_{rw}X_{wj}) + q_{rv} - Q_{raj}\right) \\ + \max\left(0, \displaystyle\sum_{\substack{w \in N_a^V \\ w \neq v}} (q_{rw}X_{wi}) - q_{rv} - Q_{rai}\right)\end{array}\right) \tag{19}$$

$\Delta^L_{M_1(v,i,j)}$ represents the intrinsic link cost difference from $currentSol$ to $neighbourSol$, given by:

$$\begin{aligned}\Delta^L_{M_1(v,i,j)} = &\sum_{\substack{l = vw \in L_t^V \\ w \in N^V \\ w \neq v}} \left(C^L_{t\varphi_1} - C^L_{t\varphi_2}\right) \\ &+ \sum_{\substack{l = wv \in L_t^V \\ w \in N^V \\ w \neq v}} \left(C^L_{t\varphi_3} - C^L_{t\varphi_4}\right),\end{aligned} \tag{20}$$

where φ_1, φ_2, φ_3 and φ_3 are respectively defined as follows:

$$\varphi_1 = \left\{\varphi \in P_{jk_w} \mid Y^+_{l\varphi} = 1, \forall l = vw \in L^V_t, w \in N^V, w \neq v\right\}$$
$$\varphi_2 = \left\{\varphi \in P_{ik_w} \mid Y^-_{l\varphi} = 1, \forall l = vw \in L^V_t, w \in N^V, w \neq v\right\}$$
$$\varphi_3 = \left\{\varphi \in P_{k_wj} \mid Y^+_{l\varphi} = 1, \forall l = wv \in L^V_t, w \in N^V, w \neq v\right\}$$
$$\varphi_4 = \left\{\varphi \in P_{k_wi} \mid Y^-_{l\varphi} = 1, \forall l = wv \in L^V_t, w \in N^V, w \neq v\right\}$$

$Y^+_{l\varphi}$ and $Y^-_{l\varphi}$ represent respectively the new and the old assignment of VL l on the given path φ. k_w represents the CP to which VM w is assigned, i.e.:
$k_w = \left\{i \in I \mid X_{wi} = 1, \forall\ w \in N^V\right\}$

$\Delta^{P^L}_{M_1(v,i,j)}$ represents the link penalty cost difference from $currentSol$ to $neighbourSol$, given by:

$$\Delta^{P^L}_{M_1(v,i,j)} = -10^5 \times \left(\left|H^L_{neighbourSol}\right| - \left|H^L_{currentSol}\right|\right), \tag{21}$$

where $H^L_{currentSol}$ and $H^L_{neighbourSol}$ represent the set of VLs whose the resulting assignment, respectively in the current solution and the neighbour solution, does not respect the constraint (13). Note that at each movement, each VL is computed to be assigned to the shortest path (in terms of PoP networks spanned) between a chosen pair of CPs that does not violate its maximum delay.

Tabu list

To avoid repeated solutions that keep returning to a local optimum, once the best move $M_1(v, i, j)$ is applied, the Tabu list is updated by adding the pair (v, i) to $TabuList$, which forbids the heuristic to assign VM v to CP i for S^{TL} iterations. Details related to the formula we use to define the length S^{TL} of the Tabu list are given in Subsection 6.2.

5.2.4 Long-term memory mechanism

The *diversification* mechanism aims at generating new solutions from so far unexplored regions. To this end, after each iteration of the LS process, the resulting solution is kept in a statistics memory table, which stores the number of times each VM has been assigned to a CP. Then, based on the least explored assignment criterion, the *diversification* mechanism creates a new initial solution $newInitSol$ from which a new exploration restarts. An *intensification* method around the neighbourhood of the best solution has also been tested in this work, but the *diversification* mechanism has given us far better solutions for such an optimisation problem.

6 Numerical results

In this section, the efficiency of the proposed TS-based VNRs splitting strategy is evaluated through simulations. To this end, two main series of experiments with different scenarios considered in each of them are conducted: *experiment 1*, where the performance of *TS_Split* algorithm is evaluated in terms of quality of solutions obtained and computing time; and *experiment 2*, where the performance of the proposed VNRs splitting strategy is evaluated according to several performance metrics, such as acceptance rate, execution time, splitting rate and delay.

The proposed approach is implemented in C++ under Visual Studio 14. All simulations are run on a single server with an Intel Core i7 CPU at 4 GHz and 32 GB of RAM.

Table 1 Experiments parameters

	Value/distribution interval	
Substrate network		
Number of nodes per CP	25	
Number of links per CP	50 on average	
Degree of nodes interconnectivity per CP	[3, 6]	
Intra-cloud link delay bound	[0, 3] ms	
Inter-cloud link / PoP transit delay bound	[0, 25] ms	
	Small sized VNRs	*Large sized VNRs*
Data centre CPU capacity per node category	[100, 500] cores	[250, 750] cores
Data centre memory capacity per node category	[1,000, 5,000] GB	[2,500, 7,500] GB
Data centre disk capacity per node category	[10,000, 50,000] GB	[25,000, 75,000] GB
Intra-cloud link bandwidth capacity per link type	[4,000, 5,000] Mbps	[6,000, 8,000] Mbps
Virtual network request		
VNR average arrival rate	1 per 100 time units	
VNR lifetime	[1,000 100,000] time units	
VL maximum delay allowed	[0, 300] ms	
	With 5 CPs	*With 10 CPs*
VM CPU demand	[1, 20] cores	[1, 40] cores
VM memory demand	[1, 200] GB	[1, 400] GB
VM disk demand	[1, 2,000] GB	[1, 4,000] GB
VL bandwidth demand	[1, 20] Mbps	[1, 20] Mbps

6.1 Experiments setup

Each of *experiment 1* and *experiment 2* is conducted considering both the strategy with 5 CPs and 10 CPs. We have also stated two scenarios in each experiment, respectively for small sized instances and large sized instances.

- *Experiment 1:* Here we first consider a *scenario 1* which compares the proposed *TS_Split* algorithm with the exact method. The latter, named *Exact_Split*, is used to calculate the optimal solution of the proposed ILP and is implemented in AMPL 64 bits with the CPLEX 12.6.3 solver. In *scenario 1*, we consider 14 instances of small sized VNRs, in which the number of VMs increases with an incremental 5-step, respectively from 5 to 70. *TS_Split* algorithm is executed 10 times for each instance. Then, the proposed algorithm is evaluated for large sized VNRs in a *scenario 2*, by also running 10 times *TS_Split* for 18 instances of VNRs, in which the number of VMs increases with an incremental 25-step, respectively from 75 to 500. In this first experiment, we compare results obtained by also studying different cases where the *diversification* mechanism and the aspiration criterion are not performed in the proposed algorithm.
- *Experiment 2:* Here we evaluate the efficiency of our VNRs strategy according to the performance metrics listed above. To this end, we implement in C++ the splitting approach proposed by Leivadeas et al. (2013), which we name *ILS_Split*. Our algorithm is performed there with the aspiration criterion and the long-term memory. In a *scenario 3*, we compare *Exact_Split*, *TS_Split* and *ILS_Split* approaches with small sized VNRs (with 5 to 70 VMs), by running 50 simulations with 100 random incoming requests in each simulation. In a *scenario 4*, we compare only *TS_Split* and *ILS_Split* approaches with large sized VNRs (with 75 to 500 VMs) that the exact method could not solve in polynomial time. We run there 10 simulations with 50 random incoming requests in each simulation. In order to evaluate the acceptance rate metric, for each of *Exact_Split*, *TS_Split* and *ILS_Split* approach, we use the same adopted method of Larumbe and Sanso (2013) for the intra-cloud mapping phase, presented in Appendix. The adopted approach is implemented for *scenario 3* with the exact method using the CPLEX solver. For *scenario 4*, we implement in C++ an iterated descent algorithm as LS method to perform it.

All topologies and features of the substrate network and the VNRs are randomly generated by using a MATLAB program and the parameters set in Table 1. Three types of computational resources were considered, which are CPU, memory and disk. VNRs topologies are randomly generated in a partial mesh, with 50% probability of interconnection between VMs. The multi-cloud substrate network interconnecting all participating CPs is generated based on ISPs topologies (Bhamare et al., 2015; Doverspike et al., 2010). Data centres in each substrate intra-cloud are interconnected through a backbone network, similar to the NSFNet topology (Amokrane et al., 2013). Substrate nodes are randomly located at different geographic areas and each data centre is connected to the backbone network through the closest routers to its location. The number of substrate nodes in each CP is set to 25, with 20% probability of generating data centre nodes and 80% of router nodes. We consider that data centres are heterogeneous (can support different categories of node) with different resource capacities. Residual resource capacities for substrate nodes and substrate links, as well as resources advertised in the discovery framework, are dynamically updated by CPs after a VNR has been mapped or an existing VNR has been released. Details related to the setup of cost parameters used to execute the mapping phase can be consulted in Appendix.

6.2 Parameters of TS_Split

Preliminary tests performed on small and large instances allowed us to define the optimal parameters of the proposed algorithm, which mostly depend on the problem size. The parameter S^{TL} is defined as $1/3 * \sum_{a \in A} \left|N_a^V\right| * |I_a|$, where I_a represents the set of CPs supporting nodes of category $a \in A$. For small sized VNRs (*scenario 1* and *scenario 3*), the parameters N^{max} and D^{max} are defined as follows: $N^{max} = 40 * \sqrt{|N^V| * |I|}$ and $D^{max} = 25 * \sqrt{|N^V| + |I|}$. For large sized VNRs (*scenario 2* and *scenario 4*), $N^{max} = 20 * \sqrt{|N^V| * |I| / 2}$ and $D^{max} = 5 * 10^4 * \sqrt{|I| / |N^V|^3}$. The number of restarts of the *Diversification* mechanism D^{max} is reduced for large instances of the problem due to the fact that the cost improvement is negligible compared to the high increase in the resulting CPU time. Indeed, as VMs are interconnected in partial mesh, with n VMs the number of VLs (in both directions of communication) increases considerably in the range of $\frac{n(n-1)}{2}$. The space of feasible solutions is then significantly reduced because of the higher probabilities of resulting in some violated VLs than in the case of small sized VNRs. Therefore, it becomes useless for the heuristic to explore new areas more than necessary.

6.3 Results analysis

The results obtained with each of *experiment 1* and *experiment 2* are presented in the following sections.

6.3.1 TS_Split with experiment 1

Tables 2, 3, 4 and 5 show the results obtained with *scenario 1*, each of them considering the algorithm with the long-term memory and without. For each VNR instance, we report the optimal cost and the CPU time given by the exact method. To demonstrate the efficiency of the proposed heuristic, we present the cost gaps (minimal, maximal and mean) with the optimal solution, expressed in percentage and calculated as $100 * (cost(TS\_Split) - cost(Exact\_Split))/cost(Exact\_Split)$. We also report the minimal, maximal and mean CPU execution time of the algorithm.

The reported results indicate on average zero cost gaps for VNRs with less than 50 VMs with the long-term memory executed. Table 2 presents the comparison between *Exact_Split* and *TS_Split* with 5 CPs, where the aspiration criterion is performed. From the table, we can see that *TS_Split* algorithm is able to reach quasi-always the optimal solution when the long-term memory is performed. The minimal cost gap is between 0% and 0.1% and the maximal one is below 1.95%, which shows that the proposed heuristic is very effective in finding optimal or near-optimal solutions. However, cost gaps are higher when the long-term memory is not performed, giving minimal and maximal cost gaps up to 10.31% and 25% respectively. This proves that the *diversification* mechanism significantly improves the quality of the solutions, allowing the heuristic to effectively explore more promising areas of research. In Table 3, which presents the results of comparison with 5 CPs with no aspiration criterion performed, we can observe on average a slight deterioration in the quality of the solutions, with a maximal cost gap up to 2.69% when the long-term memory is executed. With no *diversification* mechanism, the cost gap is more significant without the aspiration criterion, giving maximums of gap up to 28%.

Same conclusions are noticed in Tables 4 and 5. *TS_Split* is as effective when the complexity of the problem is increased with 10 CPs. As shown in Table 4, with the long-term memory and the aspiration criterion performed, most of the cost gaps are below 0.4%, with an average of gap between 0% and 2.97%. There too, with no aspiration criterion, as shown in Table 5, the cost gap increases a little, giving an average of gap up to 4.04%. With no *diversification* mechanism, the quality of solutions is considerably degraded, giving a maximal cost gap up to 32.1% and 34.3%, respectively with and without the aspiration criterion performed.

The analysis of the execution time shows that the CPU time is correlated and increases on average according to the size of the instances. With the exact method, as expected, the CPU time increases exponentially from 0.07 s with 5 VMs to 7,846.1 s with 70 VMs in the scenario with 5 CPs, and from 5.6 s with 5 VMs to 49,743.2 s with 70 VMs in the scenario with 10 CPs. The proposed heuristic significantly reduces the CPU time which becomes linear, besides being effective in finding near-optimal solutions. As shown in Tables 2 and 4, with the long-term memory we can respectively observe CPU time averages of only 0.09 s up to 35.39 s with 5 CPs, and from 0.34 s up to 78.83 s with 10 CPs. With no aspiration criterion performed, Tables 3 and 5 show CPU times a little more reduced, with averages ranging from 0.07 s to 34.69 s and from 0.25 s to 69.85 s, respectively with 5 CPs and 10 CPs. This is due to the fact that without aspiration criterion, in each iteration the heuristic only evaluates non-Tabu moves, which is less expensive in terms of calculation time, however we have solutions slightly of lower quality. Regarding all the four tables, without the long-term memory, the CPU times are very low, with a maximum up to 0.29 s and 0.8 s respectively with 5 CPs and 10 CPs, but at the price of much less good quality of solutions.

Results obtained with *scenario 2* are presented in Tables 6, 7, 8 and 9. For such large sized instances of the problem, the *Diversification* mechanism was necessary mostly because of the high number of VLs the heuristic has to satisfy. The simple algorithm without long-term memory can easily be trapped in bad local optima, sometimes leading to non-feasible solutions due to some violated VLs. In each of these four tables, for each instance, we report first the minimal, maximal and mean evaluation cost. Then, we present the gap between the mean cost and the maximal cost found by the heuristic. Also, among the 10 simulations that could reach the best solution found by the heuristic, we report the average of the total number of iterations $nbIter$ that was necessary, as well as the average of the iterations $bestIter$ where the best solution was found. The minimal, maximal and mean execution time is also given.

As shown in Tables 6 and 8, the algorithm is effective even for large sized instances, giving averages of solution cost very close to the maximal values found. Indeed, the average gap for all instances between the mean and the maximal cost is about 0.16% and 0.17%, respectively with 5 CPs and 10 CPs. In addition, the mean $bestIter$ observed, regarding the total number $nbIter$, shows that on average the algorithm can converge fast towards the best solution found. On the other hand, as expected, the execution time is higher with the large cases of the problem, but still satisfactory given the complexity of the scenarios that makes harder to effectively explore all the solution space. With 5 CPs, we can note averages of CPU time ranging from 36.02 s for 75 VMs to 608.36 s for 500 VMs. With 10 CPs, the mean CPU time increases naturally from 84.69 s to 1,765.91 s, respectively for instances from 75 VMs to 500 VMs.

Table 2 Comparison between *TS_Split* and *Exact_Split* with 5 CPs and aspiration criterion performed

Exact_Split			*TS_Split with diversification*						*TS_Split with no diversification*					
Number of VMs	*Cost*	*CPU (s)*	*Cost gaps (%)*			*CPU (s)*			*Cost gaps (%)*			*CPU (s)*		
			Min	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*
5	7.17	0.07	0	0	0	0.05	0.1	0.09	0	0	0	0	0	0
10	17.25	0.48	0	0	0	0.08	0.15	0.14	0	0	0	0	0	0
15	23.58	20.9	0	0	0	0.23	0.31	0.3	0	0.94	0.18	0	0	0
20	35.13	228.8	0	0	0	0.47	0.59	0.56	0	1.13	0.71	0	0.01	0
25	47.27	405.1	0	0	0	0.93	1.08	1.06	0	2.68	1.25	0.01	0.01	0.01
30	37.88	711.9	0	0	0	1.49	1.73	1.62	0	3.5	1.56	0.01	0.01	0.01
35	58.54	1,635.3	0	0	0	2.92	3.15	3.03	0	5.36	2.12	0.02	0.04	0.02
40	75.1	3,147.6	0	0	0	4.95	5.14	5.07	0.05	4.65	2.08	0.03	0.04	0.03
45	62.17	3,862.2	0	0	0	6.86	7.08	7.04	1.16	8.55	3.47	0.04	0.1	0.05
50	61.73	4,134.3	0	0.64	0.18	10.09	10.64	10.36	2.21	10.9	5.49	0.06	0.09	0.07
55	87.69	4,400.6	0	1.11	0.24	14.48	14.96	14.74	3.61	14.07	7.74	0.08	0.13	0.1
60	86.94	5,801.9	0.01	0.81	0.21	19.96	20.12	20.4	4.23	19.89	9.52	0.11	0.16	0.12
65	91.89	5,347.8	0.06	1.74	0.5	25.95	26.27	26.03	10.31	22.45	11.66	0.13	0.16	0.14
70	76.2	7,846.1	0.1	1.95	1.17	35.17	35.41	35.39	9.74	25.02	11.92	0.17	0.29	0.2

Table 3 Comparison between *TS_Split* and *Exact_Split* with 5 CPs and no aspiration criterion performed

Number of VMs	*Exact_Split*		*TS_Split with diversification*						*TS_Split with no diversification*					
	Cost	*CPU (s)*	*Cost gaps (%)*			*CPU (s)*			*Cost gaps (%)*			*CPU (s)*		
			Min	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*
5	7.17	0.07	0	0	0	0.03	0.1	0.07	0	0	0	0	0	0
10	17.25	0.48	0	0	0	0.08	0.13	0.11	0	0	0	0	0	0
15	23.58	20.9	0	0	0	0.18	0.3	0.25	0	1.07	0.29	0	0	0
20	35.13	228.8	0	0	0	0.36	0.58	0.44	0	1.43	0.72	0	0.01	0
25	47.27	405.1	0	0	0	0.7	1.12	0.78	0	2.83	1.78	0	0.01	0.01
30	37.88	711.9	0	0	0	1.11	1.45	1.16	0.01	3.73	2.26	0.01	0.01	0.01
35	58.54	1,635.3	0	0	0	2.14	2.2	2.15	0.03	5.88	2.92	0.01	0.02	0.02
40	75.1	3,147.6	0	0.04	0.01	3.84	4.19	4.01	0.05	6.02	3.25	0.02	0.03	0.03
45	62.17	3,862.2	0	0.03	0.02	6.37	6.91	6.28	1.17	9.43	4.1	0.03	0.06	0.04
50	61.73	4,134.33	0	0.63	0.34	9.66	9.95	9.67	2.34	12.01	5.56	0.04	0.08	0.05
55	87.69	4,400.6	0.01	1.16	0.9	13.65	13.87	13.68	3.61	15.7	7.9	0.07	0.14	0.1
60	86.94	5,801.9	0.01	0.9	0.82	18.87	19.12	18.41	4.29	28.01	11.61	0.05	0.2	0.12
65	91.89	5,347.8	0.06	1.92	0.64	24.14	24.81	24.62	10.31	24.3	13.49	0.11	0.18	0.13
70	76.2	7,846.01	0.18	2.69	1.55	34.49	34.88	34.69	9.76	27.1	14.05	0.17	0.21	0.18

Table 4 Comparison between *TS_Split* and *Exact_Split* with 10 CPs and aspiration criterion performed

	Exact_Split		*TS_Split with diversification*						*TS_Split with no diversification*					
Number	*Cost*	*CPU (s)*	*Cost gaps (%)*			*CPU (s)*			*Cost gaps (%)*			*CPU (s)*		
of VMs			*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*
5	8.13	5.6	0	0.01	0	0.3	0.58	0.34	0	0.3	0.11	0	0	0
10	13.22	56.3	0	0	0	0.44	0.81	0.47	0	0.73	0.38	0	0.01	0.01
15	22.79	364.7	0	0	0	0.72	1.04	0.75	0	1.35	0.4	0.01	0.05	0.03
20	24.9	898.2	0	0	0	1.12	1.47	1.26	0	2.08	1.03	0.03	0.09	0.06
25	32.3	2,000	0	0.06	0	2.23	2.75	2.34	0	3.9	1.35	0.06	0.13	0.07
30	36.05	2,068.1	0	0.08	0	3.93	4.31	4.06	0.04	3.96	1.49	0.09	0.16	0.1
35	43.79	3,722.2	0	0.12	0	6.62	7.05	6.71	0.12	4.35	2.97	0.08	0.17	0.11
40	50.51	5,382.2	0	0.23	0.06	11.53	11.95	11.68	0.16	7.47	2.99	0.09	0.19	0.1
45	55.76	8,703.1	0.01	0.34	0.15	17.3	17.86	17.55	1.08	10.09	5.83	0.11	0.25	0.13
50	70.87	1,2705.5	0	1.65	0.37	22.07	22.63	22.39	1.47	11.29	6.94	0.2	0.23	0.19
55	78.89	19,159	0.01	1.9	0.78	34.21	35.12	34.62	2.19	16.85	9.07	0.22	0.31	0.3
60	123.76	19,470.5	0.12	2.3	2.64	44.03	45.23	44.5	5.55	22.06	11.58	0.22	0.41	0.39
65	108.9	24,818	0.22	3.15	2.5	57.98	58.44	58.09	9.65	23.35	14.92	0.29	0.57	0.4
70	101.65	49,743.2	0.44	4.13	2.97	78.29	78.96	78.83	18.04	32.1	19.64	0.4	0.8	0.53

Table 5 Comparison between *TS_Split* and *Exact_Split* with 10 CPs and no aspiration criterion performed

Exact_Split			*TS_Split with diversification*						*TS_Split with no diversification*					
Number of VMs	*Cost*	*CPU (s)*	*Cost gaps (%)*			*CPU (s)*			*Cost gaps (%)*			*CPU (s)*		
			Min	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*	*Min*	*Max*	*Mean*
5	8.13	5.6	0	0	0	0.16	0.57	0.25	0	0.31	0.12	0	0	0
10	13.22	56.3	0	0	0	0.33	0.68	0.36	0	0.75	0.41	0	0.01	0.01
15	22.79	364.7	0	0.24	0.07	0.54	0.96	0.67	0.04	1.49	0.4	0.01	0.03	0.02
20	24.9	898.2	0	0.21	0.09	1.15	1.39	1.18	0	2	1.03	0.01	0.07	0.04
25	32.3	2,000	0	0.25	0.12	1.99	2.34	2.05	0	4.13	1.41	0.02	0.1	0.05
30	36.05	2,068.1	0	0.47	0.16	3.2	3.51	3.19	0.05	4.01	1.5	0.02	0.12	0.07
35	43.79	3,722.2	0	0.43	0.24	5.19	6.16	5.21	0.11	5.38	3.01	0.05	0.15	0.1
40	50.51	5,382.2	0.01	0.31	0.18	9.34	10.12	9.54	0.18	8.85	3.57	0.08	0.16	0.09
45	55.76	8,703.1	0.01	1.48	0.27	14.23	14.72	14.41	1.08	13.16	6.13	0.1	0.17	0.11
50	70.87	12,705.5	0.06	3.76	0.58	18.24	19.47	18.58	1.89	12.48	8.15	0.18	0.2	0.14
55	78.89	19,159	0.02	4.12	1.19	28.61	29.43	28.77	2.34	17.2	9.57	0.2	0.25	0.21
60	123.76	19,470.5	0.43	5.01	2.9	36.71	38.68	37.07	5.57	23.18	11.97	0.19	0.33	0.24
65	108.9	24,818	0.56	5.83	3.08	48.06	49.14	48.59	9.74	25.08	15.83	0.34	0.41	0.38
70	101.65	49,743.2	0.5	6.91	4.84	68.45	69.36	69.85	18.67	34.3	20.6	0.3	0.45	0.42

Table 6 *TS_Split* results for large sized VNRs with 5 CPs and aspiration criterion performed

Number of VMs	*TS_Split with diversification*								
	Cost			*Mean gap to*	*nbIter (mean) #*	*bestIter (mean) #*	*CPU (s)*		
	Min	*Max*	*Mean*	*max cost (%)*			*Min*	*Max*	*Mean*
75	151.02	151.34	151.15	0.13	5,750	1,273	35.42	37.66	36.02
100	200.45	201.75	201.67	0.04	5,735	931	37.17	42.59	38.98
125	250.06	255.1	254.85	0.1	5,946	1,781	38.03	47.23	41.36
150	307.43	310.96	309.9	0.34	5,592	174	42.3	49.55	44.41
175	360.39	362.89	362.08	0.22	5,948	1,600	46.36	52.86	48.37
200	359.15	361.05	360.83	0.06	6,046	2,291	50.27	56.32	52.43
225	406.35	408.31	407.92	0.1	6,608	2,058	51.04	61.03	58.26
250	439.77	440.8	440.32	0.11	6,766	4,766	60.12	71.74	66.44
275	474.16	475.14	474.62	0.1	6,786	4,166	81.07	102.24	92.06
300	491	494.06	493.4	0.13	6316	3,034	107.33	119.47	112.6
325	504.26	507.85	506.9	0.19	6,179	1,163	136.84	141.42	139.11
350	555.16	557.58	556.74	0.15	5,922	2,021	159.14	178.64	163.82
375	539.18	541.98	540.03	0.36	6,135	1,606	202.35	210.04	206.82
400	589.2	592.24	591.08	0.2	6,315	1,385	253.82	264.11	258.64
425	632.09	636.2	635.13	0.17	6,560	1,482	319.07	359.09	334.21
450	597.94	602.18	602	0.03	6,095	2,074	360.74	405.41	371.91
475	649	652.91	650.88	0.31	6,285	1,738	458.19	470.31	464.14
500	688.23	691.74	690.17	0.23	6,330	1,381	568.32	633.94	608.36

Table 7 *TS_Split* results for large sized VNRs with 5 CPs and no aspiration criterion performed

	TS_Split with diversification								
Number of VMs	*Cost*			*Mean gap to*	*nbIter (mean) #*	*bestIter (mean) #*	*CPU (s)*		
	Min	*Max*	*Mean*	*max cost (%)*			*Min*	*Max*	*Mean*
75	149.84	151.3	150.85	0.32	5,984	3,618	34.64	37.12	35.82
100	198.59	201.75	200.05	0.84	5,692	1,900	35.1	43.18	37.58
125	250	255.1	252.5	1.02	5,985	3,514	37.83	46.9	40.96
150	305.2	310.96	307.8	1.02	5,592	174	42.01	49.11	43.21
175	360.39	362.63	360.41	0.68	5,948	1,600	46.36	52.86	48.37
200	354.15	361.05	357.91	0.87	6,046	2,291	50.27	56.32	52.43
225	400.66	408.07	405.34	0.73	6,608	2,058	50.94	60.13	57.62
250	421.41	439.68	428.83	2.72	6,824	3,824	60.07	70.87	64.58
275	455.16	475.02	466.88	1.73	6,368	2,176	80.76	92.42	84.96
300	491.03	494.06	493.27	0.16	6,187	2,030	103.89	117.27	106.96
325	500.12	507.14	505.39	0.48	6,252	1,578	133.52	141.27	136.85
350	551.37	555.95	555.88	0.3	5,915	1,424	155.24	166.68	159.92
375	530.99	541.98	537.2	0.88	6,153	2,481	199.69	206.11	203.08
400	583.24	589.07	585.39	1.16	6,409	1,605	248.25	276.69	257.85
425	603.82	636.2	629.72	1.02	6,666	1,784	317.29	336.35	328.09
450	575.75	602.18	598.43	0.62	6,252	2,902	349.69	379.97	369.18
475	628.49	652.21	643.76	1.4	6,144	1,872	439.75	450.44	444.03
500	675.4	691.74	687.95	0.55	6,857	4,170	552.19	579.69	574.99

Table 8 *TS_Split* results for large sized VNRs with 10 CPs and aspiration criterion performed

	TS_Split with diversification								
Number of VMs	*Cost*			*Mean gap to*	*nbIter (mean) #*	*bestIter (mean) #*	*CPU (s)*		
	Min	*Max*	*Mean*	*max cost (%)*			*Min*	*Max*	*Mean*
75	102.92	102.95	102.94	0.01	11,321	4,587	84.37	85.4	84.69
100	139.92	139.92	139.92	0	11,133	6,037	89.57	90.25	89.89
125	169.35	170.11	170.03	0.05	11,475	1,175	89.79	91.13	90.56
150	204.82	204.82	204.82	0	11,515	1,559	96.2	98.42	97.61
175	242.02	242.97	242.39	0.24	11,295	3,139	98.38	99.95	99.18
200	275.94	277.94	276.16	0.64	11,461	7,164	99.25	102.7	100.58
225	301.33	306.54	305.6	0.31	11,540	3,668	116.08	124.39	120.41
250	325.26	336.25	334.73	0.45	12,098	5,311	176.42	187.29	179.89
275	375.65	380.05	379.09	0.25	11,676	1,302	216.1	230.7	221.04
300	418.76	436.2	434.87	0.3	12,281	3,960	304.06	327.79	311.32
325	441.97	444.16	444.02	0.03	12,866	5,411	383.57	428.49	405.36
350	450.43	452.95	452.43	0.11	11,954	2,256	446.87	513.39	461.05
375	506.57	507.78	507.41	0.07	12,210	4,705	572.15	640.47	594.39
400	515.79	518.4	516.83	0.3	12,987	4,583	717.89	803.43	766.97
425	600.13	601.97	601.94	0	12,482	3,641	868.12	961.85	897.87
450	643.49	646.28	646.01	0.04	12,539	3,249	1,058.32	1,198.71	1,109
475	653.28	676.34	674.29	0.3	12,427	5,999	1,327.05	1,404.47	1,369.52
500	710.62	711.98	711.82	0.02	12,865	8,465	1,693.68	1,848.32	1,765.91

Table 9 *TS_Split* results for large sized VNRs with 10 CPs and no aspiration criterion performed

	TS_Split with diversification								
Number of VMs	*Cost*			*Mean gap to*	*nbIter (mean) #*	*bestIter (mean) #*	*CPU (s)*		
	Min	*Max*	*Mean*	*max cost (%)*			*Min*	*Max*	*Mean*
75	100.48	102.95	101.86	1.06	11,321	4,587	72.68	74.3	72.34
100	139.07	139.92	139.73	0.14	11,133	6,037	76.94	78.63	77.37
125	163.27	170.11	167.76	1.38	11,475	1,175	79.25	83.07	82.67
150	179.75	204.82	201.69	1.53	11,515	1,559	88.57	90.71	90.54
175	240.81	242.18	241.93	0.43	11,295	3,139	93.09	94.25	94.53
200	271.02	277.94	275.92	0.73	11,461	7,164	96.12	99.59	97.83
225	300.87	304.46	303.68	0.93	12,006	7,651	113.64	121.36	118.16
250	321.6	336.25	327.38	2.64	12,365	7,063	166.5	178.29	173.22
275	372.28	379.74	375.43	1.22	11,758	3,755	202.86	224.8	210.06
300	407.25	436.2	426.54	2.21	12,682	7,419	291.54	330.06	305.35
325	433.2	442.36	441.65	0.57	13,406	6,152	359.21	428.42	399.82
350	450.01	450.13	450.08	0.63	12,462	6,053	435.19	500.46	472.4
375	500.38	507.78	502.19	1.1	13,789	6,861	559.1	623.49	589.42
400	504.72	515.89	511.75	1.28	12,497	4,451	644.94	749.72	669.28
425	576.52	600.74	596.86	0.85	12,845	7,225	853.69	931.81	979
450	635.25	646.28	643.48	0.43	12,505	3,025	995.55	1,113.31	1,069.45
475	644.05	676.28	665.16	1.65	11,976	8,080	1,200.96	1,273.03	1,230.36
500	696.86	710.12	702.08	1.39	12,678	6,678	1,450.24	1,778.22	1,591.96

Results obtained in Tables 7 and 9 confirm that without the aspiration criterion, less good qualities of solution are noticed, with an average gap for all instances between the mean and the maximal cost of about 0.92% and 1.12%, respectively with 5 CPs and 10 CPs. Furthermore, by observing the mean $bestIter$, without the aspiration criterion it seems harder for the algorithm to converge more easily towards the best solution found. However, the execution time is reduced, giving on average from 35.82 s to 574.99 s with 5 CPs, and from 72.34 s to 1,591.96 s with 10 CPs.

The results analysis resulting from the *experiment 1* shows that the proposed algorithm is effective in solving such a combinatorial networking problem, even without the aspiration criterion performed, by offering a good tradeoff between the quality of the solutions and the highly reduced computing time.

6.3.2 TS_Split with experiment 2

The performance of the proposed VNRs splitting strategy is first evaluated with the acceptance rate, which represents one of the most conclusive metrics to evaluate the efficiency of a successful performance-based VNRs splitting strategy. The acceptance rate is measured as the number of VNRs that are successfully mapped by all CPs selected by the splitting strategy, divided by the total number of incoming VNRs. In our experiment, if only one VM or one VL of a request has a failed embedding (including the failed inter-cloud connections), we consider the entire VNR rejected. From Figure 5, which presents the average acceptance rate per size of request for small sized VNRs (i.e., with *scenario 3*), it can be seen that the acceptance rate decreases on average according to the number of communicating VMs. This is primarily due to the fact that available resources of CPs can be limited over time because of requests that have not yet expired. Thus, on average it becomes harder to satisfy all the QoS constraints for VNRs with a greater number of VMs. Our splitting strategy, with both the exact and the proposed heuristic approaches, leads to higher acceptance rates than the approach of Leivadeas et al. (2013). Indeed, the mean acceptance rate considering all requests is about 89.64% with 5 CPs and 95.09% with 10 CPs with *Exact_Split*, and about 89.34% with 5 CPs and 94.92% with 10 CPs with *TS_Split*. The latter leads to a little more requests rejections than the exact method, but almost negligible, which proves that our heuristic is as effective as the exact method to perform good decisions in the splitting strategy. However, *ILS_Split* approach is more subject to VNRs rejections, even with the large instances of the problem. For small sized VNRs as depicted in Figure 5, *ILS_Split* gives a mean acceptance rate for all requests of about 75.78% with 5 CPs and 84.15% with 10 CPs. For large sized VNRs with *scenario 4*, as illustrated in Figure 6, *TS_Split* gives a mean acceptance rate for all requests of about 88.29% with 5 CPs and 95.1% with 10 CPs, while with *ILS_Split* we note 74.65% of mean acceptance rate with 5 CPs and about 83.97% with 10 CPs. This means that with *TS_Split* strategy, we improve on average the acceptance rate by approximately 15.34% and 15.76%, respectively for small and large sized VNRs. This can be explained by the fact that *ILS_Split* strategy tends to load most of the resource demands to CPs having the largest numbers of data centres and substrate links. This may not be a representative performance criterion if data centres are heterogeneous with different capacities. Also, the bandwidth capacities on substrate paths are not necessarily related to the number of substrate links they consider in the link provisioning cost, which can results in more rejected VLs.

Figure 5 Average acceptance rate for small sized VNRs

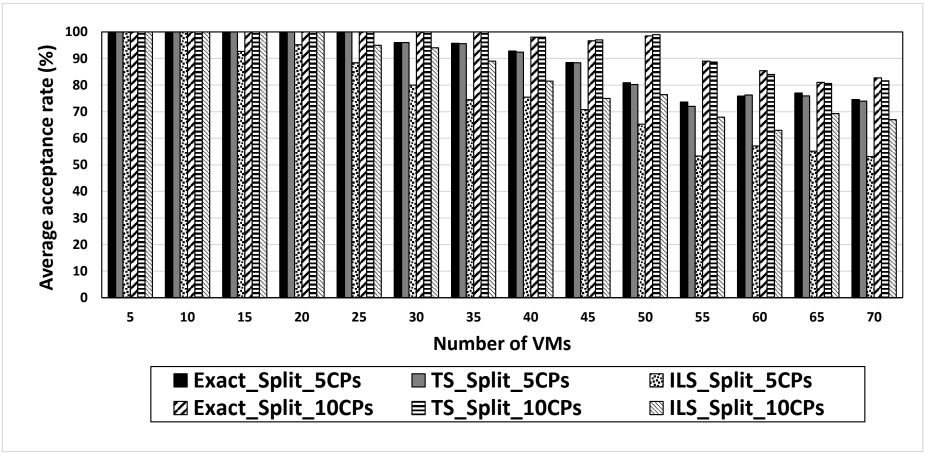
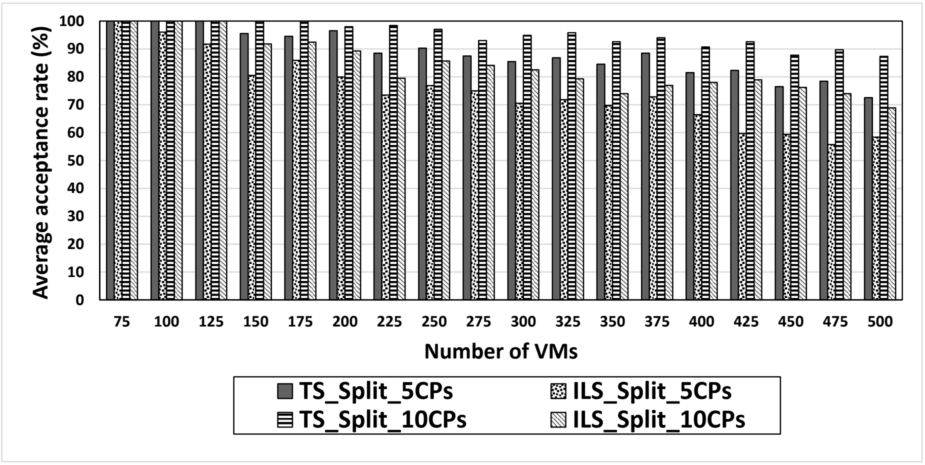


Figure 6 Average acceptance rate for large sized VNRs



The execution time results are presented in Tables 10 and 11, respectively for *scenario 3* and *scenario 4*. For small sized VNRs in Table 10, the execution time with *Exact_Split* is naturally exponential, as the solver CPLEX uses *branch-and-bound* methods. Considering all instances, in comparison with *TS_Split* which gives an average of about 9.22 s and 20.86 s respectively with 5 CPs and 10 CPs, *ILS_Split* takes a little more time to compute the splitting phase, resulting in an average of about 13.92 s and 33.4 s respectively with 5 CPs and 10 CPs. Same results can be noticed in Table 11 with large sized VNRs. The CPU time with *TS_Split* is on average 180.4 s and 488.57 s respectively with 5 CPs and 10 CPs, while with *ILS_Split* it is about 268.35 s and 778.23 s respectively with 5 CPs and 10 CPs. This is probably due to the large number of iterations they assess for the LS, and the high perturbation strength of 80% they perform at each iteration for such a combinatorial problem, leading the LS to iteratively calculate almost all the objective function.

Table 10 Average computing time for small sized VNRs

Number of VMs	*CPU (s) with 5 CPs*			*CPU (s) with 10 CPs*		
	Exact_Split	*TS_Split*	*ILS_Split*	*Exact_Split*	*TS_Split*	*ILS_Split*
5	0.09	0.14	0.11	5.2	0.4	2.04
10	0.9	0.19	0.18	63.76	0.69	4.04
15	40.93	0.32	0.59	438.39	0.94	4.92
20	200.37	0.6	0.67	885.77	1.13	6.89
25	394.7	1.18	0.85	1,664.68	2.79	10.04
30	912.5	1.7	2.12	2,485.97	4.35	12.48
35	1,836.31	3.45	4.61	3,811.83	7.37	15.13
40	3,043.93	5.43	6.68	6,307.88	12.01	18.04
45	3,688.16	7.15	10.24	7,746.59	17.24	25.05
50	4,123.48	11.01	19.36	12,326.89	23.8	36.73
55	4,375.38	15.05	27.37	19,487.75	34.98	57.15
60	5,741.5	20.41	33.45	21,795	46.51	60.54
65	6,947.85	26.8	37.1	29,121.24	59.97	70.1
70	7,486.48	35.65	51.52	49,672.83	79.89	114.39

Table 11 Average computing time for large sized VNRs

Number of VMs	*CPU (s) with 5 CPs*		*CPU (s) with 10 CPs*	
	TS_Split	*ILS_Split*	*TS_Split*	*ILS_Split*
75	35.5	55.1	84.7	159.8
100	37.1	75.3	89.9	218.4
125	43.6	90.3	90.6	261.9
150	52.4	118.8	97.6	344.5
175	63	134.6	109.2	390.3
200	71.5	149.2	119.6	432.7
225	77.8	163.9	120.4	475.3
250	85.2	175.7	179.9	509.5
275	95.8	198.1	221	574.5
300	133.9	225.4	311.3	653.7
325	140.7	238	405.4	690.2
350	162.9	274.9	461.1	797.2
375	205.6	301.7	594.4	874.9
400	255	347.5	767	1,007.8
425	328.3	422	897.9	1,223.8
450	400	512.1	1,109	1,485.1
475	467.3	643.4	1,369.5	1,865.9
500	591.6	704.4	1,765.9	2,042.8

We also evaluate the average splitting rate, calculated as the number of assigned segments to the selected CPs at each incoming VNR, divided by the total number of CPs. As shown in Figures 7 and 8, the splitting rate increases on average according to the size of requests. For small sized VNRs as depicted in Figure 7, both *Exact_Split* and *TS_Split* approaches give substantially same results, with an average splitting rate for all requests of about 51.82% and 36.73% respectively with 5 CPs and 10 CPs, while *ILS_Split* gives much higher splitting rates, which are on average 79.65% and 59.16% respectively with 5 CPs and 10 CPs. For large sized VNRs as depicted in Figure 8, the average splitting rate for all requests with *TS_Split* is about 74.73% and 51.49% respectively with 5 CPs and 10 CPs, while with *ILS_Split* it is about 97.22% and 70.83% respectively with 5 CPs and 10 CPs. This means that our strategy efficiently assigns most of the VNRs to 3 up to 5 CPs among the participating CPs, thus avoiding useless inter-cloud traffics. However, *ILS_Split* assigns most of the VNRs on average to 4 up to 7 CPs, without giving better acceptance rates. A VNR splitting rate unnecessarily high should be avoidable. It generally introduces a large number of VLs transiting through the inter-cloud paths, resulting in high overall delays and huge additional expenditures for the SP.

Figure 7 Average splitting rate for small sized VNRs

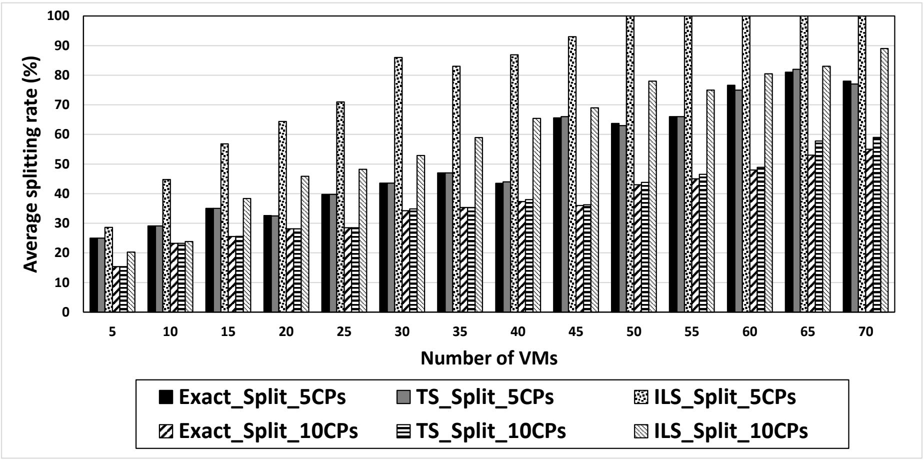


Figure 8 Average splitting rate for large sized VNRs

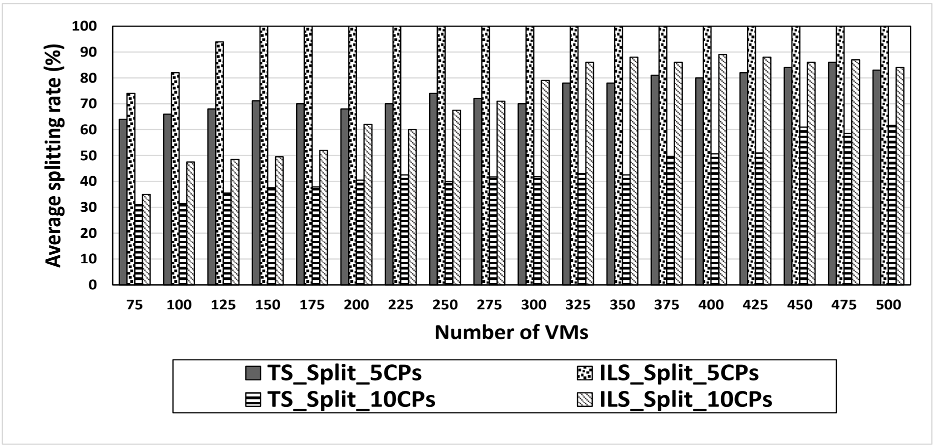


Figure 9 Average inter-cloud delay for small sized VNRs

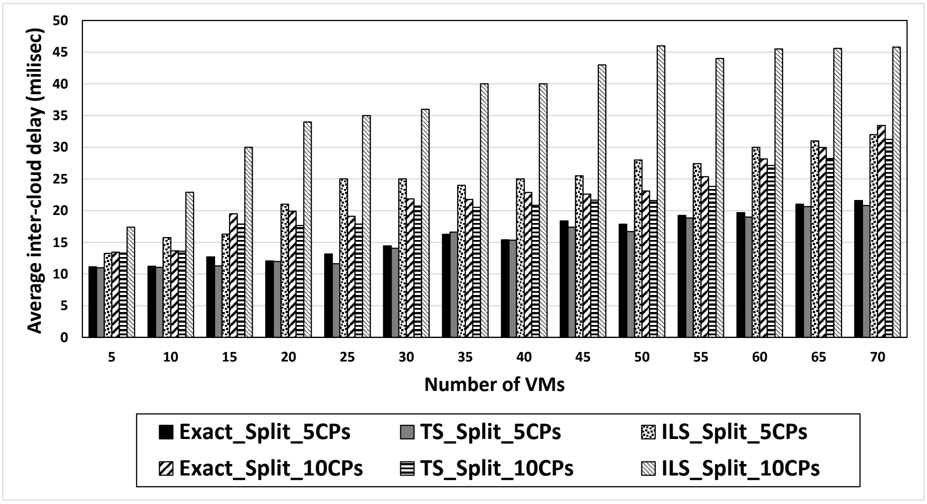
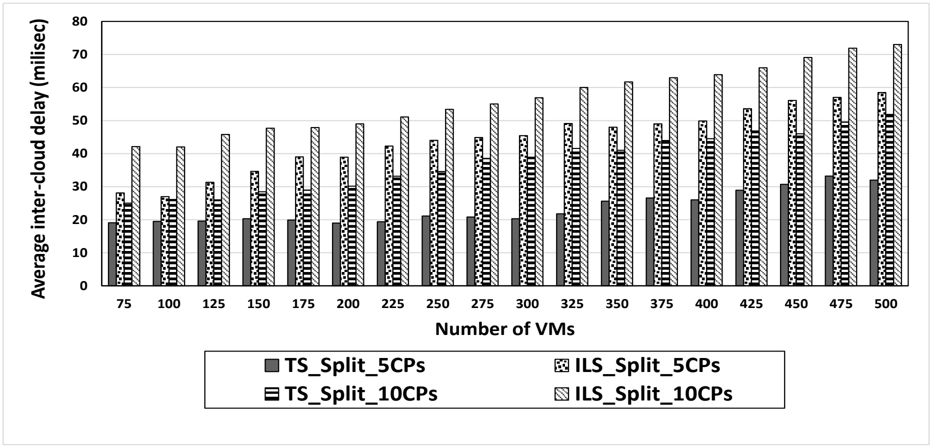


Figure 10 Average inter-cloud delay for large sized VNRs



Finally, we present the average inter-cloud delay in Figures 9 and 10, which is very closely in correlation with the longer of chosen paths and the splitting rate. The delay represents the performance metric that influences the most the QoS. As expected, for both scenarios the inter-cloud delay increases on average according to the size of requests. In Figure 9, we can see that *TS_Split* gives less inter-cloud delays than *Exact_Split* and *ILS_Split*. In comparison with the exact method, this is due to the fact that *TS_Split* algorithm is able to perform a VLs routing by considering not only the shortest path (defined by the number of hops), but also the delay on the path. The solver of exact method meanwhile randomly chooses a path between those with the same shortest length, regardless of the delay on the paths. *ILS_Split* approach gives more inter-cloud delays considering all requests, which are on average about 21.87 ms with 5 CPs and 34.93 ms with 10 CPs, against 15.45 ms with 5 CPs and 21.15 ms with 10 CPs for *TS_Split* algorithm. For large sized VNRs, Figure 10 depicts the same conclusion, with an average inter-cloud delay of about 44.27 ms with 5 CPs and 56.64 ms with 10 CPs for *ILS_Split* approach, while *TS_Split* results in an average delay of about 23.54 ms with 5 CPs and 37.51 ms with 10 CPs. This means that on average the inter-cloud delay is improved by about 40.29% with our *TS_Split* approach. We can also notice that a high splitting rate has a significant influence on the inter-cloud delay, by resulting in more VMs communicating through inter-cloud links. Thus, we can conclude that our approach penalises more effectively long inter-cloud paths than *ILS_Split* approach.

7 Conclusions

In this paper, the NP-hard problem of VNE across multiple IaaS-based cloud networks has been addressed. A TS approach including a long-term memory mechanism has been proposed in order to perform, in polynomial time, an efficient VNRs splitting strategy formalised as a mathematical ILP model. The proposed strategy aims at improving the performance and the QoS of resulting VNR segments that are mapped onto the selected cloud infrastructure networks. The comparison results with the exact method, used to execute small sized instances of the problem, show that the proposed algorithm is able to generate, in a highly reduced computing time, solution costs very close to the upper bounds, with an average cost gap from about 0% to a maximum of 2.97%.

Simulations performed with large instances of the problem and comparing our approach with an other baseline method, show the efficiency of the proposed approach in dealing with the scalability aspect of such a combinatorial networking problem. Our approach improves several performance criteria, including the acceptance rate and the network communication delay, which are respectively improved of about 15.55% and 40.29%.

Future works might be interested in developing efficient and fast dynamic hybrid metaheuristics, by combining different approaches and optimisation techniques to solve large instances of the problem. Furthermore, as the SP is also interested in minimising the resource provisioning price, it is worthwhile to consider in future works a multi-objective optimisation approach, which can extend the proposed mathematical model by also taking into account the minimisation of the SP expenditure in the splitting strategy.

References

Amazon (2017) *EC2Instance Types* [online] https://aws.amazon.com/fr/ec2/instancetypes/ (accessed 13 Dec 2017).

Amokrane, A., Zhani, M.F., Langar, R., Boutaba, R. and Pujolle, G. (2013) 'Greenhead: virtual data center embedding across distributed infrastructures', *IEEE Transactions on Cloud Computing*, Vol. 1, No. 1, pp.36–49.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010) 'A view of cloud computing', *Communications of the ACM*, Vol. 53, No. 4, pp.50–58.

Ayoubi, S., Assi, C., Shaban, K. and Narayanan, L. (2015) 'MINTED: multicast virtual network embedding in cloud data centers with delay Ccnstraints', *IEEE Transactions on Communications*, Vol. 63, No. 4, pp.1291–1305.

Belbekkouche, A., Hasan, M.M. and Karmouch, A. (2012) 'Resource discovery and allocation in network resource virtualization', *IEEE Communications Surveys and Tutorials*, Vol. 14, No. 4, pp.1114–1128.

Bhamare, D., Jain, R., Samaka, M., Vaszkun, G. and Erbad, A. (2015) 'Multi-cloud distribution of virtual functions and dynamic service deployment: open ADN perspective', in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, pp.299–304.

Chaisiri, S., Lee, B-S. and Niyato, D. (2012) 'Optimization of resource provisioning cost in cloud computing', *IEEE Transactions on Services Computing*, Vol. 5, No. 2, pp.164–177.

Chowdhury, M., Rahman, M.R. and Boutaba, R. (2012) 'Vineyard: virtual network embedding algorithms with coordinated node and link mapping', *IEEE/ACM Transactions on Networking (TON)*, Vol. 20, No. 1, pp.206–219.

Dietrich, D., Rizk, A. and Papadimitriou, P. (2015) 'Multi-provider virtual network embedding with limited information disclosure', *IEEE Transactions on Network and Service Management*, Vol. 12, No. 2, pp.188–201.

Doverspike, R.D., Ramakrishnan, K. and Chase, C. (2010) 'Structural overview of ISP networks', in Kalmanek, C., Misra, S. and Yang, R. (Eds.): *Guide to Reliable Internet Services and Applications*, pp.19–93, Springer, London.

Fischer, A., Botero, J.F., Beck, M.T., De Meer, H. and Hesselbach, X. (2013) 'Virtual network embedding: a survey', *IEEE Communications Surveys and Tutorials*, Vol. 15, No. 4, pp.1888–1906.

Glover, F. (1989) 'Tabu search – part i', *ORSA J. Computing*, Vol. 1, No. 3, pp.190–206.

Glover, F. (1990) 'Tabu search – part ii', *ORSA J. Computing*, Vol. 2, No. 1, pp.4–32.

Gong, S., Chen, J., Yin, X. and Zhu, Q. (2016) 'Survivable virtual network embedding across multiple domains', in *Computer and Communications IEEE 2nd International Conference*.

Grozev, N. and Buyya, R. (2012) 'Inter-cloud architectures and application brokering: taxonomy and survey', *Software – Practice and Experience*, Vol. 44, No. 3, pp.369–390.

Hesselbach, X., Amazonas, J.R., Villanueva, S. and Botero, J.F. (2016) 'Coordinated node and link mapping VNE using a new paths algebra strategy', *Journal of Network and Computer Applications*, Vol. 69, pp.14–26.

Houidi, I., Louati, W., Ameur, W.B. and Zeghlache, D. (2011) 'Virtual network provisioning across multiple substrate networks', *Computer Networks*, Vol. 55, No. 4, pp.1011–1023.

Houidi, I., Louati, W. and Zeghlache, D. (2015) 'Exact multi-objective virtual network embedding in cloud environments', *The Computer Journal*, Vol. 58, No. 3, pp.403–415.

Justafort, V.D., Beaubrun, R. and Pierre, S. (2015) 'On the carbon footprint optimization in an intercloud environment', *IEEE Transactions on Cloud Computing*, Vol. 6, No. 3, pp.829–842.

Khan, M.M.A., Shahriar, N., Ahmed, R. and Boutaba, R. (2016) 'Multi-path link embedding for survivability in virtual networks', *IEEE Transactions on Network and Service Management*, Vol. 13, No. 2, pp.253–266.

Larumbe, F. and Sanso, B. (2012) 'Cloptimus: a multi-objective cloud data center and software component location framework', in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, IEEE, pp.23–28.

Larumbe, F. and Sanso, B. (2013) 'A tabu search algorithm for the location of data centers and software components in green cloud computing networks', *IEEE Transactions on Cloud Computing*, Vol. 1, No. 1, pp.22–35.

Leivadeas, A., Papagianni, C. and Papavassiliou, S. (2013) 'Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, No. 6, pp.1077–1086.

Li, S., Saidi, M.Y. and Chen, K. (2016) 'A cloud-oriented algorithm for virtual network embedding over multi-domain', in *Local Computer Networks Workshops IEEE 41st Conference*.

Lv, B., Wang, Z., Huang, T., Chen, J. and Liu, Y. (2010) 'Virtual resource organization and virtual network embedding across multiple domains', in *Multimedia Information Networking and Security (MINES), 2010 IEEE International Conference*, pp.725–728.

Mano, T., Inoue, T., Ikarashi, D., Hamada, K., Mizutani, K. and Akashi, O. (2016) 'Efficient virtual network optimization across multiple domains without revealing private information', *IEEE Transactions on Network and Service Management*, Vol. 13, No. 3, pp.477–488.

Manvi, S.S. and Shyam, G.K. (2014) 'Resource management for infrastructure as a service (IaaS) in cloud computing: a survey', *Journal of Network and Computer Applications*, Vol. 41, pp.424–440.

Mechtri, M., Hadji, M. and Zeghlache, D. (2015) 'Exact and heuristic resource mapping algorithms for distributed and hybrid clouds', *IEEE Transactions on Cloud Computing*, Vol. 5, No. 4, pp.681–696.

Melo, M., Sargento, S., Killat, U., Timm-Giel, A. and Carapinha, J. (2013) 'Optimal virtual network embedding: node-link formulation', *IEEE Transactions on Network and Service Management*, Vol. 10, No. 4, pp.356–368.

Melo, M., Sargento, S. Killat, U., Timm-Giel, A. and Carapinha, J. (2015) 'Optimal virtual network embedding: energy aware formulation', *Computer Networks*, Vol. 91, No. C, pp.184–195.

Rafael, M-V., Rubén, S.M. and Ignacio, M.L. (2012) 'IaaS cloud architecture: from virtualized datacenters to federated cloud infrastructures', *IEEE Computer Society*, Vol. 45, No. 12, pp.65–72.

Rahman, M.R. and Boutaba, R. (2013) 'Svne: Survivable virtual network embedding algorithms for network virtualization', *IEEE Transactions on Network and Service Management*, Vol. 10, No. 2, pp.105–118.

Samuel, F., Chowdhury, M. and Boutaba, R. (2013) 'Polyvine: policy-based virtual network embedding across multiple domains', *Journal of Internet Services and Applications*, Vol. 4, No. 1, pp.1–23.

Sanchis, L. (1989) 'Multiple-way network partitioning', *IEEE Trans. Computers*, Vol. 38, No. 1, pp.62–81.

Tao, L., Zhao, C., Thulasiraman, K. and Swamy, M. (1992) 'Simulated annealing and Tabu search algorithms for mulitway graph partitioning', *Journal of Circuits, Systems and Computers*, Vol. 2, No. 2, pp.159–185.

Zhang, S., Qian, Z., Wu, J., Lu, S. and Epstein, L. (2014) 'Virtual network embedding with opportunistic resource sharing', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 3, pp.816–827.

Zhang, J., Huang, H. and Wang, X. (2016) 'Resource provision algorithms in cloud computing: a survey', *Journal of Network and Computer Applications*, Vol. 64, No. C, pp.23–42.

Appendix

Formulation of the VNR segments intra-cloud mapping problem

In this appendix, we present the MILP formulation of the adopted intra-cloud mapping approach, which follows the work of Larumbe and Sanso (2013). Details related to the setup of cost parameters used for our simulations are also given. All the notation used for the modelling of the intra-cloud substrate network and the VNR segments, as well as the formulation of the problem, can be consulted in Table 12.

Table 12 Notation for the VNR segments intra-cloud mapping problem

Symbols	*Description*
Global sets	
A	Set of node categories
T	Set of link types
R	Set of computational resource types
I	Set of CPs
Substrate intra-cloud network	
G_i^S	Graph representing the substrate network of CP i
N_i^S	Set of substrate nodes in G_i^S
L_i^S	Set of substrate links in G_i^S
D_i	Set of data centres of CP i
ϕ_i	PoP node (transit network) of CP i
$\mathcal{F}_i$	Set of all paths in CP i's network
$\mathcal{P}_{dc}$	Set of all paths between data centres d and c, $d, c \in D_i$
$\mathcal{P}_{d\phi_i}$	Set of all paths between data centre $d \in D_i$ and PoP node ϕ_i
$\mathcal{K}_e$	Set of all paths in $\mathcal{F}_i$ spanning link $e \in L_i^S$
B_{te}	Available bandwidth capacity of the channel of type $t \in T$ of link $e \in L_i^S$
d_e	Delay bound of link $e \in L_i^S$
Q_{rad}	Available capacity of resource $r \in R$ in data centre $d \in D_i$ for nodes of category $a \in A$ ($Q_{rad} \in \mathbb{N}$)
U_{adr}	Usage of resource $r \in R$ by all VMs of node category $a \in A$ assigned to data centre $d \in D_i$
E_{adr}	Average power (in watts) consumed by a VM of node category $a \in A$ assigned to data centre $d \in D_i$ in terms of resource $r \in R$
ω_d	Average power (in watts) consumed by all VMs assigned to data centre $d \in D_i$
ρ_d	Power Usage Effectiveness (PUE) of data centre $d \in D_i$
θ_d	CO_2 emissions in data centre $d \in D_i$ (in g/KWh)
Virtual network request segment	
G_i^V	Graph representing the VNR segment assigned to CP i
N_i^V	Set of VMs assigned to CP i
L_i^V	Set of VLs assigned to CP i
N_{ai}^V	Set of VMs of node category $a \in A$ assigned to CP i
L_{ti}^V	Set of VLs of type $t \in T$ exclusively assigned to CP i
$L_{t\phi_i}^V$	Set of VLs of type $t \in T$ assigned to an inter-cloud link of endpoint CP i
q_{rv}	Amount of resource $r \in R$ required by VM $v \in N_i^V$
b_l	Bandwidth demand of VL $l \in L_i^V$
δ_l	Maximum delay allowed for VL $l \in L_i^V$
Costs	
c_{rai}^S	Unit resource cost for CP i for using resource $r \in R$ for nodes of category $a \in A$ (in \$/unit)
c_{ti}	Unit bandwidth cost for CP i for using a link channel of type $t \in T$ (in \$/Mbps)
$c_{ti}^{\mathcal{E}}$	Extra unit bandwidth cost for CP i for using a link channel of type $t \in T$ (in \$/Mbps)

Table 12 Notation for the VNR segments intra-cloud mapping problem (continued)

Symbols	*Description*
Costs	
c_d^{ω}	Unit electricity cost in data centre $d \in D_i$ (in \$/MWh)
$\wp^{\mathcal{D}}$	Penalty for each millisecond of delay on VLs (in \$/ms)
$\wp_d^{\mathcal{O}}$	Penalty for emitting CO$_2$ in data centre $d \in D_i$ (in \$/tonne)
$C_i^{\mathcal{S}}$	Total computing resource cost for CP i (in \$/h)
$C_i^{\mathcal{T}}$	Total traffic cost for CP i (in \$/h)
$C_i^{\mathcal{D}}$	Total delay penalty for CP i (in \$/h)
C_i^{ω}	Total energy cost for CP i (in \$/h)
$C_i^{\mathcal{O}}$	Total environmental penalty for CP i (in \$/h)
Decision variables	
X_{vn}	Binary variable set to **1** if VM $v \in N_{ai}^V,\ a \in A$, is assigned to substrate node $n \in N_i^S$; **0** otherwise
$Y_{l\varphi}$	Binary variable set to **1** if VL $l \in L_{ti}^V,\ t \in T$, is assigned to path $\varphi \in \mathcal{P}_{dc}$, $d, c \in D_i$; **0** otherwise
$Z_{k\gamma}$	Binary variable set to **1** if VL $k \in L_{t\phi_i}^V,\ t \in T$, is assigned to path $\gamma \in \mathcal{P}_{d\phi_i}$, $d \in D_i$; **0** otherwise

Adopted intra-cloud mapping approach

The following are the formulas that define each cost and penalty of the multi-objective function. In our solution, the mapping of a VM remains at the data centre level. VMs requirements are expressed in terms of computational resource demand in set R, not in terms of number of servers. Therefore, some formulas originally defined by Larumbe and Sanso (2013) were re-stated. Moreover, we did not consider the path splitting scenario, neither the data centres CAPEX and OPEX since we assign VMs to existing data centres. Note that each cost is defined in dollars per hour (\$/h).

Servers cost is re-stated as computational resources cost and is calculated as follows:

$$C_i^{\mathcal{S}} = \sum_{d \in D_i} \sum_{r \in R} \sum_{a \in A} \sum_{v \in N_{ai}^V} c_{rai}^{\mathcal{S}} q_{rv} X_{vd} \tag{22}$$

The total traffic cost for CP i is defined as follows, with an extra cost stated for inter-cloud VLs:

$$\begin{aligned} C_i^{\mathcal{T}} = & \sum_{d \in D_i} \sum_{c \in D_i} \sum_{\varphi \in \mathcal{P}_{dc}} \sum_{e \in \varphi} \sum_{t \in T} \sum_{l \in L_{ti}^V} c_{ti} b_l Y_{l\varphi} \\ & + \sum_{d \in D_i} \sum_{\gamma \in \mathcal{P}_{d\phi_i}} \sum_{e \in \gamma} \sum_{t \in T} \sum_{k \in L_{t\phi_i}^V} \left(c_{ti} + c_{ti}^{\mathcal{E}}\right) b_k Z_{k\gamma} \end{aligned} \tag{23}$$

The total delay penalty for CP i is re-defined as follows:

$$C_i{}^{\mathcal{D}} = \wp^{\mathcal{D}} \left(\begin{aligned} & \sum_{d \in D_i} \sum_{c \in D_i} \sum_{\varphi \in \mathcal{P}_{dc}} \sum_{e \in \varphi} \sum_{t \in T} \sum_{l \in L_{ti}^V} \delta_e Y_{l\varphi} \\ & + \sum_{d \in D_i} \sum_{\gamma \in \mathcal{P}_{d\phi_i}} \sum_{e \in \gamma} \sum_{t \in T} \sum_{k \in L_{t\phi_i}^V} \delta_e Z_{k\gamma} \end{aligned} \right) \tag{24}$$

The average power consumed by all VMs assigned to data centre $d \in D_i$ is re-stated as follows, estimated as a function of the utilisation of CPU resource and disk resource, by following the principles used by Justafort et al. (2015):

$$\omega_d = \sum_{a \in A} \left(E_{dar_1} U_{dar_1} + E_{dar_2} U_{dar_2} \right), \forall\ d \in D_i, \tag{25}$$

where E_{dar_1} and E_{dar_2} represent (in watts) the average power consumed by a VM of node category $a \in A$ assigned to data centre d, respectively in terms of CPU resource and disk resource. U_{dar_1} and U_{dar_2} define respectively the CPU resource usage and the disk resource usage by all VMs of node category $a \in A$ assigned to data centre d, given by:

$$U_{dar} = \frac{\sum_{v \in N_{ai}^V} q_{rv} X_{vd}}{\max\left(Q_{rad}, 1\right)}, \quad \forall\ d \in D_i,\ a \in A,\ r \in R \tag{26}$$

The total energy cost for CP i is then defined as follows:

$$C_i^{\ \omega} = 10^{-6} \sum_{d \in D_i} c_d^{\omega} \rho_d \omega_d, \tag{27}$$

The total environmental penalty for CP i is given as follows:

$$C_i^{\ \mathcal{O}} = 10^{-9} \sum_{d \in D_i} \wp_d^{\mathcal{O}} \theta_d \rho_d \omega_d \tag{28}$$

The objective function for the intra-cloud mapping phase is then defined as follows, with each cost weighted by a parameter that allows CPs to modify the costs priority:

MIN

$$\alpha \mathcal{C}_i^{\mathcal{S}} + \beta \mathcal{C}_i^{\mathcal{T}} + \lambda \mathcal{C}_i^{\mathcal{D}} + \varpi \mathcal{C}_i^{\omega} + o \mathcal{C}_i^{\mathcal{O}} \tag{29}$$

Subject to:

$$\sum_{n \in N_i^S \setminus D_i} X_{vn} = 0, \quad \forall\ v \in N_{ai}^V,\ a \in A \tag{30}$$

$$\sum_{d \in D_i} X_{vd} = 1, \quad \forall\ v \in N_{ai}^V,\ a \in A \tag{31}$$

Constraints (30) and (31) ensure respectively that VMs are only assigned to data centres, and each VM must be assigned to exactly one data centre.

$$Y_{l\varphi} \le \frac{X_{ud} + X_{vc}}{2}, \quad \forall\ l = uv \in L_{ti}^V,\ t \in T, u, v \in N_{ai}^V (v \neq u),\ a \in A,\ \varphi \in \mathcal{P}_{dc},\ d, c \in D_i \tag{32}$$

$$Z_{k\gamma} \le X_{vd}, \quad \forall\ k = v\phi_i \in L_{t\phi_i}^V,\ t \in T,\ v \in N_{ai}^V,\ \gamma \in \mathcal{P}_{d\phi_i},\ d \in D_i \tag{33}$$

Constraints (32) and (33) state respectively the binary value of variables $Y_{l\varphi}$ and $Z_{k\gamma}$.

$$\sum_{d\in D_i}\sum_{c\in D_i}\sum_{\varphi\in\mathcal{P}_{dc}} Y_{l\varphi}=1, \quad \forall\, l\in L^V_{ti},\ t\in T \tag{34}$$

$$\sum_{d\in D_i}\sum_{\gamma\in\mathcal{P}_{d\phi_i}} Z_{k\gamma}=1, \quad \forall\, k\in L^V_{t\phi_i},\ t\in T \tag{35}$$

Constraint (34) ensures that each VL exclusively assigned to a CP must be mapped to a unique path between two different data centres, otherwise to a unique path intra-data centre. Constraint (35) ensures that each VL partially assigned to a CP must be mapped to a unique path between a data centre and the PoP node of the CP.

$$\sum_{v\in N^V_{ai}} q_{rv}X_{vd}\le Q_{rad}, \quad \forall\, r\in R,\ a\in A,\ d\in D_i \tag{36}$$

$$Q_{rad}=Q_{rad}-\sum_{v\in N^V_{ai}} q_{rv}X_{vd}, \quad \forall\, r\in R,\ a\in A,\ d\in D_i \tag{37}$$

Constraint (36) ensures that the total amount of a resource required by all VMs assigned to a data centre must not exceed its residual capacity. Constraint (37) updates this residual capacity after each VNR is successfully mapped.

$$B_{\overrightarrow{te}}=B_{\overleftarrow{te}}, \quad \forall\, e\in L^S_{ti} \tag{38}$$

Constraint (38) states that the bandwidth capacity of a channel of a link type is the same in both directions on every substrate link.

$$\sum_{m\in N^S_i} f^{uv}_{nm}+X_{vn}b_l=\sum_{m\in N^S_i} f^{uv}_{mn}+X_{un}b_l, \quad \forall\, u,v\in N^V_i,\ v\ne u,\quad n\in N^S_i,\ l=uv\in L^V_i \tag{39}$$

Constraint (39) guaranties the flow conservation for every amount of traffic $l=uv$ from VM u to VM v, with a bandwidth demand b_l routed on substrate link $e=mn\in L^S_i$.

$$\sum_{\varphi\in\mathcal{K}_e}\sum_{l\in L^V_{ti}} b_lY_{l\varphi}+\sum_{\gamma\in\mathcal{K}_e}\sum_{k\in L^V_{t\phi_i}} b_kZ_{k\gamma}\le B_{te}, \quad \forall\, e\in L^S_i,\ t\in T \tag{40}$$

$$B_e=B_e-\left(\sum_{\varphi\in\mathcal{K}_e}\sum_{l\in L^V_{it}} b_lY_{l\varphi}+\sum_{\gamma\in\mathcal{K}_e}\sum_{k\in L^V_{t\phi_i}} b_kZ_{k\gamma}\right), \quad \forall\, e\in L^S_{ti},\ t\in T \tag{41}$$

Constraint (40) ensures that the total bandwidth demand of all VLs routed on a substrate link must not exceed the residual bandwidth capacity of the corresponding channel. Constraint (41) updates this residual capacity after a VNR is successfully mapped.

$$\sum_{d\in D_i}\sum_{c\in D_i}\sum_{\varphi\in\mathcal{P}_{dc}}\sum_{e\in\varphi} d_eY_{l\varphi}\le\delta_l, \quad \forall\, l\in L^V_{ti},\ t\in T \tag{42}$$

$$\sum_{d\in D_i}\sum_{\gamma\in\mathcal{P}_{d\phi_i}}\sum_{e\in\gamma} d_eZ_{l\gamma}\le\delta_k, \quad \forall\, k\in L^V_{t\phi_i},\ t\in T \tag{43}$$

Constraints (42) and (43) ensure that the restriction on the maximum delay allowed for a VL is not violated.

$$X_{vn} \in \{0,1\} \quad \forall\, v \in N^V_{ai},\ a \in A,\ n \in N^S_i \tag{44}$$

$$Y_{l\varphi} \in \{0,1\} \quad \forall\, l \in L^V_{ti},\ t \in T,\ \varphi \in \mathcal{P}_{dc},\ d,c \in D_i \tag{45}$$

$$Z_{k\gamma} \in \{0,1\} \quad \forall\, k \in L^V_{t\phi_i},\ t \in T,\ \gamma \in \mathcal{P}_{d\phi_i},\ d \in D_i \tag{46}$$

$$\omega_d \in \mathbb{R}_{\geq 0} \quad \forall\, d \in D_i \tag{47}$$

Constraints (44)–(47) express the domain of definition of each variable.

Cost parameters setup

The cost parameters setting follows the experimentation setup defined in Larumbe and Sanso (2013), but they are specified for an hour time period. We chose the delay as the first optimisation priority of the multi-objective function, by using the same weight attribution as in Larumbe and Sanso (2013). The unit computational resource cost is randomly determined in the interval \$[50, 60]/cores/h for CPU resource, \$([50, 60])/10/GB/h for memory resource and \$([50, 60])/100/GB/h for disk resource. The unit bandwidth cost for each link type is randomly chosen in the interval \$[12, 15]/Mbps/h, and the corresponding extra unit bandwidth cost is estimated at 25% of the initial unit cost. The delay penalty is set to \$0.15/(ms/packet)/h. The average power consumed by a VM assigned to a data centre is randomly set in the interval [200, 300] W (with 60% of the energy for CPU resource and 40% for disk resource). The unit electricity cost is randomly distributed in the interval \$[30, 70]/MWh. The PUE of a data centre is 1.5. The amount of CO_2 emissions in a data centre is calculated by summing values in the set {10, 66, 443, 960} g/KWh, each of them first multiplied by a greenness factor randomly set between 0 and 1. The penalty for emitting CO_2 in a data centre is defined as \$1,000/tonne.