Optimizing resource utilization in NFV dynamic systems: New exact and heuristic approaches

Thi-Minh Nguyen, Michel Minoux, Serge Fdida

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128618302287
Manuscript_e6ca38aeb657c17b8a95792faff94bdd

Optimizing Resource Utilization in NFV Dynamic Systems: New Exact and Heuristic Approaches

Thi-Minh Nguyen[a], Michel Minoux[a], Serge Fdida[a]

[a]*Sorbonne Université, UPMC, LIP6, CNRS UMR 7606, France*

Abstract

Network Function Virtualization (NFV) orchestration and management have attracted a lot of attention in recent years as it provides new opportunities regarding performance and deployment. In particular, several models have attempted to capture the behavior of such systems under various restricted assumptions. However, previously proposed mathematical models can only handle problems of fairly small size. This paper proposes a Mixed Integer Linear Programming (MILP) model for the resource utilization problem in an NFV dynamic context with several enhancements regarding the state of the art. We include the utilization of flow constraints to ensure the order of functions in a service chain. By systematic generation of Flow Cover inequalities, significant improvements in processing time are obtained with a standard MILP solver to compute exact optimal solutions. We also propose three efficient heuristics (two MILP-based heuristics) to find high-quality feasible solutions for large-scale systems within reduced execution time. We also carry out a set of experiments to evaluate the proposed algorithms and provide valuable guidelines for the efficient design of such systems. The results show that our approach is capable of handling large size instances of the NFV deployment problem involving up to 200 nodes and 100 demands.

Keywords: Network Function Virtualization (NFV), Resource allocation, Network performance analysis, NFV dynamic system, Flow Covers, Placement

1. Introduction

Thanks to virtualization technology, Network Function Virtualization (NFV) offers a new way to design, deploy and manage a network and its services. NFV decouples the network functions, such as Firewall and Load Balancing from proprietary hardware appliances and moves them to virtual servers. These functions can be managed within a network on demand and scaled up and down as needed, without the delay and cost of installing new hardware devices. However in case of high demands, some resources end up being over-utilized, resulting in higher latency and SLA degradation, whilst for low demands waist of resources may occur. In such circumstances and in order to meet the performance and energy objectives, the Vitual Network Function (VNF) instances need to be dynamically located on the network. Hence, the placement of VNF is a central issue in the NFV deployment stage. An essential step in connection with this issue is to develop efficient tools for NFV placement and routing, to optimize the resource utilization on both nodes and links.

NFV placement has been extensively investigated in the literature as reported e.g. in the survey papers [1], [2]. Proposed mathematical models take into account various constraints such as the amount of available resources on nodes and links or the order of VNFs in a chain. However, in order to keep the model tractable, most existing works rely on simplifying important assumptions. For instances, they consider that a set of possible paths between any pair of nodes is known or that the placement of functions is fixed. More importantly, the solution methods available in the existing literature are only capable of handling small size instances.

The present paper provides a mathematical model to the joint problem of VNFs placement and routing path selection, for chaining them upon service request

Email addresses: thi-minh.nguyen@lip6.fr (Thi-Minh Nguyen), michel.minoux@lip6.fr (Michel Minoux), serge.fdida@upmc.fr (Serge Fdida)

from users. The routing paths will be steered through a number of VNFs, with the goal of executing the network service in the required order of functions. Our model uses flow constraints to ensure that the functions in a service chain are processed in the right order whilst keeping the model simple and tractable. We note here that, the principle of flow constraints has already been suggested in [3], [4] but in these references, the MILP models do not appear to lend themselves to efficient exact solutions. Moreover, the applications investigated in [3], [4] significantly differ from those addressed in the present paper. In [3], the allocation of **a specific Service Chaining** applying load balancing is considered. For each time step, they run the model to solve with only one connection (one demand). In [4], the general resource allocation problem for user requests in cloud computing is considered, but without taking into account the placement of each VNF on the substrate network. We also note that in contrast with [3] and [4], the approach of the present work includes the admission control decision of Service Providers either to serve a demand or to reject it.

The main contributions of the present paper are the following:

- We define and formulate the static version of the resource utilization problem as a Mixed Integer Linear Programming (MILP) problem. A distinctive feature of our model contrasting with most existing work is that the model is linear and can find the optimal solution for configurations of fairly large size. Thus the model can also be applied in the context of operating a dynamic system in order to minimize the resource utilization and maximize the number of accepted demands.

- We propose an innovative method for generating some valid inequalities (Flow Covers) to improve efficiency of the solution process. The method allows us to report on experiments involving large instances of the problem. To the best of our knowledge, this is the first time Flow Cover inequalities are used in the context of optimizing NFV systems.

- We propose three powerful heuristics, where there are two MILP-based heuristics, to find accurate feasible solutions for larger instances of the problem.

- We evaluate the proposed model and solution methods on realistic network topology using an event-driven simulation to run our model in a dynamic scenario. We also evaluate the distribution of VNFs on a new data centre architecture (a Leaf-Spine topology) and suggest that VNFs should be located on Leaf-Layer rather than a Spine-Layer.

The rest of the paper is organized as follows. In Section 2, we discuss the relevant literature and contributions. Section 3 presents our MILP model. The improvement in solution efficiency via generation of Flow Cover inequalities is in Section 4. Three heuristic algorithms for approximately solving larger instances are proposed in Section 5 and in Section 6. Results of simulation carried out on a number of network instances (up to 200 nodes, 100 demands and 5 functions) in Section 7. Finally, conclusions and perspective are presented in Section 8.

2. Overview of Related Work

The problem of resource allocation in NFV Infrastructure has been extensively studied and surveys of models and solutions method are provided in [1], [2]. The placement problem is known to be NP Complete [5] and various mathematical models have been proposed in [6], [7], [8], [9], [10] and [11]. In particular, in [6], Lukovszki *et al.* formulated the offline (SCEP: Service Chain Embedding Problem) and online problems (OSCEP: Online SCEP). They assumed that a list of all potential paths, that can be used for routing through VNFs chaining for each demand, is available.

Li et *.al* [7] and Papagianni [4] addressed the optimal allocation of Virtual Resources in Cloud Computing Networks. They proposed methods for general efficient mapping of user requests (virtual resources) to a shared substrate interconnecting network. They did not take into account the placement of VNFs on the substrate network. In [8], Elias *et al.* formulated the centralized version of VNF-Forwarding Graph Embedding (VNF-FGE) as a non-linear integer optimization model assuming that the placement of functions is fixed. They derived the best solution for each individual Virtual Operator. In other words, these model solved with only one request (one demand) for each running time. Also, as above mentioned, in [3], Leivadeas proposed the allocation model of a specific Service Chain applying load balancing.

Recently, Sun *et al.* [11] investigated the offline and online solutions for the VNF placement problem with the aim of minimizing the deployment cost. The chaining of VNFs is taken into account but a pre-defined set of paths for routing the demands is assumed.

Ma *et al.* [12] proposed a non-linear model for the network function virtualization problem depending on the dynamic requirements of the network but do not solve their model exactly. An algorithm based on Min-Max routing algorithm is proposed. Liu *et al.* [13] also jointly optimized the deployment of new users' Service Function Chains (SFCs) and the readjustment of in-service users' SFCs while considering the trade-off between resource consumption and operational overhead. They formulated a path-based ILP model to solve the problem assuming that a set of predefined paths between any pair of nodes is known and given. An approximate algorithm based on Column Generation (CG) model is also developed but the approach can only solve small instances such as 6-nodes topology and 14-nodes NSFNET topology.

In [14], Addis *et al.* defined the generic VNF chain routing optimization problem and proposed a mixed integer linear programming formulation. Their model took into consideration specific NFV forwarding models (standard and fast path modes) as well as flow bit-rate variations that make the allocation of edge demands over VNF chains unique yet complex. However, they did not solve the model exactly but only approximately via a math-heuristic resolution method.

Recently, Kuo et al. [15] studied the joint problem of VNF placement and path selection to better utilize the network. They studied the relation between the path length and the virtual machine reuse factor. However, they only proposed a chain deployment heuristic to find a solution whose path length and reuse factor approximately meet given target values.

Leivadeas *et al.* [3] studied the problem of deploying service chains on an SDN enabled data center network. A mixed integer linear programming model is proposed but the latter does not appear to lend itself to efficient resolution using standard MILP solvers, and no exact solution is reported in the paper. Heuristic algorithms are used to recalculate the routing paths in order to adjust to dynamic traffic. Moreover, the size of instances used in their computational experiments are fairly small (a fat tree topology $k = 4$ with 28 nodes).

In connection with the resource distribution problem in cloud radio access network, Hui *et al.* [16], [17] studied the multi-dimensional resources integration (MDRI) for service provisioning. The proposed architecture was experimentally verified on OpenFlow-based enhanced SDN testbed in terms of resource utilization, path blocking probability, network cost, and path provisioning latency. However, they are different from the original NFV placement problem when requests are only considered as network flows transferring from source to destination with the needed network and processing resources, but without the order requirement of nodes on network paths.

In contrast to existing works, the present paper provides a new efficient MILP formulation for the static version of NFV placement and routing problem considering the order of VNF in a chaining. A novel feature of this formulation is the systematic generation of valid inequalities referred to as Flow Cover inequalities, thus greatly improving the efficiency of the solution process. Our model does not restrict routing to a predetermined set of paths. The model not only can find an exact optimal solution to large instances of the static version of the problem but can also be applied in the analysis of a dynamic scenario by taking into account the decision of Service Providers to serve a demand or reject it. For handling instances of larger size, for which finding exact optimal solution would be computationally too expensive, several heuristic solution algorithm are proposed to provide satisfactory trade-offs between solution qualify and computational efficiency.

3. Problem Formulation

3.1. System Description

In our system, we distribute resources of the physical network to virtualized network functions. Each VNF must be mapped to a physical node equipped with sufficient resources to host the VNF. In addition, the logical links between VNFs must be mapped to physical paths so that all physical links on the paths have sufficient bandwidth to accommodate the logical links.

We model a network as an edge-weighted vertex-weighted directed graph $G = (V, E)$, in which $w_{u,v}$ is the link capacity of each edge (link) $(u, v) \in E$ and c_v is the number of resources (i.e. virtual machines

(VMs)) that can be hosted on each vertex (server) $v \in V$.

A virtual network function is defined through various characteristics such as the input data rate that it has to handle and its output interfaces, based on the forwarding rules. In this paper, we assume that the data rate of the flows does not change when processed through network functions. Hence, a VNF is characterized by its processing requirement (q_f) for each $f \in F$.

A demand (or request) d is defined by a source s^d, a destination t^d, the bandwidth requirement b^d, a chaining of ordered VNFs (F^d) through which the flow has to be processed in the right order.

In this paper, we focus on solving the dynamic problem characterized by the fact that demands can occur at any time. In particular, in the situation of heavy load, it is critical to provide a model capable to decide which demand will be served and how to allocate efficiently resources to demands. After a time period δ (depending on the system configuration), we collect demands that arrive in the time slot $[t-\delta, t[$, where t is the current time. We also update the available system resources to take into account possible completion of demands in this time period. We call $D^{(t)}$ the set of demands arriving between $t-\delta$ and t. $G^{(t)} = (V^{(t)}, E^{(t)})$ is defined a the current state of network G at time t. The current network state at time t includes the set of available resources of the system after time $t-\delta$ and the set of resources released by demands completed during the period $[t-\delta, t[$.

We aim to optimize the location of functions and demand routing in a typical time interval $[t-\delta, t[$. This is the basic sub-problem to be solved in any time steps for operating a complex dynamic system. The goal of the formulation is to obtain an efficient placement of VNFs and routing of the flows without violating the constraints induced by the available resources on nodes and links. The objective function aims at minimizing the utilization of the links and nodes, thus maximizing the number of accepted demands.

3.2. Mixed Integer Linear Programming Formulation

3.2.1. Notation

We define the outputs of our problem as a set of decision variables.

- x^d_{fv}: a binary variable that equals to 1 if and only if node v hosts function f of demand d
- y^d_{uv}: a binary variable that equals to 1 if and only if demand d uses link (u, v)
- $\varphi^{f_i,d}_{u,v}, i = 0..l_d + 1$ equals to 1 if and only if (u, v) is used in a path between two successive functions f_{i-1}, f_i for demand d, where l_d is the number of functions required by demand d. Especially, $\varphi^{f_0,d}_{u,v}$ represents the case of a path between the source and the first function f_0, whilst $\varphi^{f_{l_d+1},d}_{u,v}$ stands for the case of a path between the last function f_{l_d} and the destination.
- z^d equals to 1 if and only if demand d is served

We aim at minimizing the maximum resource utilization and maximize the acceptance ratio. The resource utilization includes the utilization on nodes and links.

$$U^{(t)} = \alpha * \frac{\sum_{d \in D^{(t)}} z^d}{|D^{(t)}|} - \beta * (\mathcal{L}^{(t)} + \mathcal{N}^{(t)}) \qquad (3\text{–}1)$$

where:

$$\mathcal{L}^{(t)} = \underset{(u,v)\in E^{(t)}}{Max} \left\{ \frac{\sum_{d \in D^{(t)}} b^d y^d_{uv}}{w_{uv}} \right\} \qquad (3\text{–}2)$$

$$\mathcal{N}^{(t)} = \underset{v \in V^{(t)}}{Max} \left\{ \frac{\sum_{d \in D^{(t)}, f \in F^d} q_f x^d_{fv}}{c_v} \right\} \qquad (3\text{–}3)$$

The choice of the two design parameters α and β influences the way our system serves customers. The basic principle is that we want to serve as many demands as possible (the first term) or we will reject demands that will consume too much resources proportionally (the second term, including the resources on nodes and links). For example, in our simulation, with $|D^{(t)}| = 100$ we use $\alpha = 10$ and $\beta = 1$ to assure that a demand is served if the amount of resources it needs is reasonable (typically less than 10% of the available resources on both nodes and links of the system).

3.2.2. Mathematical Model

($\mathcal{P}$): **Maximize** $U^{(t)}$

Subject to:

$$\sum_{d\in D^{(t)}, f\in F^d} x^d_{fv} q_f \le \mathcal{N} c_v \quad \forall v \in V^{(t)} \tag{3–4}$$

$$\sum_{d\in D^{(t)}} y^d_{uv} b^d \le \mathcal{L} w_{uv} \quad \forall u, v \in V^{(t)} \tag{3–5}$$

$$y^d_{uv} + y^d_{vu} \le 1 \quad \forall u, v \in V^{(t)}, d \in D^{(t)} \tag{3–6}$$

$$\sum_{v\in V^{(t)}} x^d_{fv} = \begin{cases} z^d & \text{if } f \in F^d \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D^{(t)} \tag{3–7}$$

$$\sum_{v\in V^{(t)}} y^d_{uv} - \sum_{v\in V^{(t)}} y^d_{vu} = \begin{cases} z^d & \text{if } u = s^d \\ -z^d & \text{if } u = t^d \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D^{(t)} \tag{3–8}$$

$$\varphi^{df}_{uv} \le y^d_{uv} \quad \forall u, v \in V^{(t)}, d \in D^{(t)}, f \in F^d \tag{3–9}$$

$$\mathcal{A} * \varphi^{df_i} = x^d_{f_{i-1}} - x^d_{f_i} \quad \forall d \in D^{(t)} \tag{3–10}$$

$$y^d_{uv} \le z^d \quad \forall u, v \in V^{(t)}, d \in D^{(t)} \tag{3–11}$$

$$\varphi^{df}_{uv} \le z^d \quad \forall u, v \in V^{(t)}, d \in D^{(t)}, f \in F^d \tag{3–12}$$

$$x^d_{fv} \in \{0,1\}, y^d_{uv} \in \{0,1\}, \varphi^{df}_{uv} \ge 0 \tag{3–13}$$

Constraint (3–4), (3–5) ensure that the node capacity and link capacity are not exceeded. Constraint (3–7) states that a required function is only processed one time for each demand. Constraint (3–6) means that for each demand we avoid to transfer a demand back on a path. Constraint (3–8) expresses flow conservation at each node. Constraints (3–9) and (3–10) enforce the order of function in VNFs chaining.

In (3–10), $\mathcal{A}$ denotes the node-arc incidence matrix of the network, $x^d_{f_i}$ is defined for each demand d as an n-dimensional integral vector, n is the number of nodes and its components are $x^d_{f_i v}$. More precisely, for each demand d, $x^d_{f_{-1}}$ is the n-dimensional vector with all components be 0 except component of its source that equals to 1. Similarly, $x^d_{f_{l_d+1}}$ is the n-dimension vector with all components be 0 except component of its destination that equals to 1.

The joint effect of constraints (3–8), (3–9) and (3–10) is to guarantee that for each demand d, the corresponding functions are indeed located on the elementary path from its source to its destination in the solution. As a result, the ordering constraints will be correctly expressed.

Constraints eqs. (3–11) and (3–12) take into account the decision of Service Providers to accept or reject a demand in a dynamic system. For instance, if demand d is blocked (that means $z^d = 0$), we do not allocate any resource to this demand ($y^d_{uv} = 0$ and $\varphi^{df}_{uv} = 0$ with $\forall u, v \in V, f \in F, d \in D$).

4. Flow Cover Inequalities

The mathematical model above is a Mixed Integer Linear Programming problem that can be solved by MILP based branch-and-bound algorithms. Here, we use GUROBI 6.52 optimization software to compute the solution. However, with the academic version of Gurobi, the efficient algorithms for finding strong valid inequalities that cut off fractional solutions to MILP relaxations are limited. Moreover, without addition of valid inequalities, the relaxations are not strong enough to efficiently prune the nodes. Therefore, the problem is hard to solve for large instances. The purpose of this section is to present strong valid inequalities for the above model that can be used successfully in reducing the size of the branch-and-bound tree. Our inequalities are derived from the flow cover inequalities that were introduced by Padberg et al. [18] and Van Roy et al. [19, 20, 21].

Consider the set of feasible points for a linear integer programming given by:

$$Y = \{y^d_{uv} \in \{0,1\} : \sum_{d\in D^{(t)}} y^d_{uv} b^d \le w_{uv} \quad \forall u, v \in V^{(t)}\}$$

Here $\{h^d_{uv} = b^d * y^d_{uv}\}$ is the used bandwidth on link (u, v) of demand d. We have:

$$\begin{cases} \sum_{d\in D^{(t)}} h^d_{uv} \le w_{uv} \quad \forall u, v \in V^{(t)} \\ h^d_{uv} \le b^d * y^d_{uv} \quad y^d_{uv} \in \{0,1\}, h^d_{uv} \ge 0 \end{cases} \tag{4–1}$$

We consider (4–1) as a single-node flow model with exogenous supply w_{uv} and $|D^{(t)}|$ outflow arc (see Figure 1)

For each demand $d \in D^{(t)}$, the flow h^d_{uv} on the d^{th} arc is bounded by the capacity $b^d > 0$ if the arc is open. Note that y^d_{uv} equals 1 if (u, v) is used by

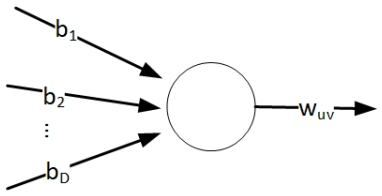


Figure 1: An example for a *single-node flow* model

demand d, and equals 0 otherwise. Hence, we have: $b^d \leq w_{uv}, \quad \forall (u,v) \in E^{(t)}$.

Definition 1. *A set S is called a flow cover, with respect to link $(u,v) \in E^{(t)}$, if $\sum_{d \in S} b^d > w_{uv}$*

Initially, we consider the subset of Y with $y^d_{uv} = \eta^d$, i.e, η^d is fixed at one of its bounds, for $d \in D^{(t)} \backslash D_0$ given by:

$$Y^0 = \{y^d_{uv} \in \{0,1\} : \sum_{d \in D_0} y^d_{uv} b^d \leq \xi \quad \forall u,v \in V^{(t)}\}$$

where $\xi = w_{uv} - \sum_{d \in D^{(t)} \backslash D_0} \eta^d b^d$

The flow cover inequalities in the context of our model are then expressed as:

$$0 \leq w_{uv} - \sum_{d \in S} y^d_{uv} b^d - \sum_{d \in S^+} (b^d - \rho)(1 - y^d_{uv}) \quad (4\text{–}2)$$

where $\rho = \sum_{d \in S} b^d - w_{uv}$, $S^+ = \{d \in S, b^d > \rho\}$

We now need to find cover sets S leading to strong inequalities of the form (4–2). We propose below a cover set selection strategy that uses knowledge from previous nodes on the branch-and-bound tree. In each new node in the branch-and-bound tree, we know fractional solutions of MILP relaxation, namely the values of y^d_{uv}, denoted by $\overline{y}^d_{uv}$.

Considering each link (u,v) successively, we know a set of values $\overline{y}^d_{uv}$ with $d \in D^{(t)}$. In order to find $S_0 \subset D^{(t)}$ such that $\sum_{d \in S_0} b^d > w_{uv}$, we sort demands in $D^{(t)}$ according to ascending order of the values $|\overline{y}^d_{uv} - \frac{1}{2}|$. Based on this ordered list, we then select a valid cover set S_0 by picking up the $k-$first demands in the list such that the sum of their bandwidth requirement exceeds w_{uv} and k is minimum. Now we have to check the violation of (4–2) with the current solution before deciding to add this cover. A flow cover inequality is added only if it is violated by the current solution $\overline{y}^d_{uv}$. Therefore, in order to increase the opportunity of finding flow cover inequalities violated by the current fractional solution, we apply an enumeration process to extend the cover set obtained by adding τ more demands from the sorted list (beginning at $(k+1)^{th}$ demand) to the current cover set S_0. We now have an extended cover set S^e having $k+\tau$ demands. Note that a cover S is valid if $\sum_{d \in S} b^d > w_{uv}$. Therefore we then enumerate all valid covers from the extended cover set S^e. As a result, we get at most $C^k_{k+\tau}$ cover sets where $k <= |D^{(t)}|$. For each cover set, the corresponding flow cover inequality is generated and appended to the model whenever it is violated by the current solution $\overline{y}$.

Lemma 1. *If a set S is a Flow Cover, selected by the above procedure, with respect to link (u,v) then the inequalities (4–2) is violated by the current solution $\overline{y}^d_{uv}$*

Proof. It follows from Definition 1 that S is a Flow Cover, we have $\sum_{d \in S} b^d > w_{uv}$ and $b^d - \rho > 0, \forall d \in S^+$. Hence, $\rho = \sum_{d \in S} b^d - w_{uv} > 0$. Assume that $\overline{y}^d_{uv}$ is the current solution of $\mathcal{P}$, clearly $\overline{y}^d_{uv} \leq 1$ and it holds that:

$$\begin{aligned}
W_{uv} &= w_{uv} - \sum_{d \in S} \bar{y}^d_{uv} b^d - \sum_{d \in S^+} (b^d - \rho)(1 - \bar{y}^d_{uv}) \\
&< w_{uv} - \sum_{d \in S} \bar{y}^d_{uv} b^d - \sum_{d \in S^+} b^d (1 - \bar{y}^d_{uv}) \\
&< w_{uv} - \sum_{d \in S \backslash S^+} \bar{y}^d_{uv} b^d - \sum_{d \in S^+} b^d \\
&< w_{uv} - \sum_{d \in S} b^d + \sum_{d \in S \backslash S^+} b^d - \sum_{d \in S \backslash S^+} \bar{y}^d_{uv} b^d \\
&< (w_{uv} - \sum_{d \in S} b^d) + \sum_{d \in S \backslash S^+} b^d (1 - \bar{y}^d_{uv}) \\
&< -\rho + \sum_{d \in S \backslash S^+} b^d (1 - \bar{y}^d_{uv})
\end{aligned}$$

Following the procedure that we pick demands to add to the cover set, we have selected demands whose $|\bar{y}^d_{uv} - 1/2|$ is near to 0, in other hand $\bar{y}^d_{uv} \approx 1/2$. With $d \in S \backslash S^+$, we have $b^d \leq \rho$. Hence, $\sum_{d \in S \backslash S^+} b^d (1 - \bar{y}^d_{uv}) < \rho/2$. So $W_{uv} < -\rho/2 < 0$ with $\rho > 0$. That violates the inequalities (4–2).

□

In our simulation, we use the optimization tool (GUROBI optimization [22]) for solving the MILP problem $\mathcal{P}$. In the implementation of our model, we do not use GUROBI's default parameter as we aim to estimate the efficiency of our proposed flow covers when compared to using the default parameters of the optimization tool. Using the Gurobi callback routines, we stop at every new node on the branch-and-bound tree (that means, at those nodes, the problem found the fractional solution and we can obtain those values), we then insert our own code to include our solution inside the optimization tool (GUROBI). In particular, we generate some strong flow covers as discussed above and add them to our current model. The optimization process continues until it can find an optimal solution. That reduces the size of the branch-and-bound tree quickly if the flow covers are strong enough. More details regarding the efficiency of our solution will be found in the computational Section 7.

In the following section, we propose three efficient heuristics to find a feasible solution within reduced computation time.

5. MILP based Heuristics

In this section, we propose two approximate solution procedures based on our MILP model.

5.1. Node-Sub-Optimal Algorithm (NOSO)

The NOde-Sub-Optimal Algorithm consists in first determining the optimal solution of the sub-problem obtained by imposing integrality on the x variables only and relaxing the other binary variables (y-variables) into continuous variables. After obtaining the solution from this relaxed version of our model, we set the x-variables at their optimal values for this sub-problem. The remaining problem is then solved to optimally enforcing integrality of the variables y. Therefore the problem space is decreased significantly. Finally, we solve this problem with those binary variables (y) by using the MILP solver GUROBI to obtain the final heuristic solution.

5.2. Path-Sub-Optimal Algorithm (PASO)

This seconde heuristic is very similar to the previous one, except for interchanging the roles of the x and y variables.

More precisely, in PASO, we consider another sub-problem by keeping the y-variables as only binary variables while relaxing x-variables as the continuous variables. The binary y-variables are then fixed at the values that we attain when dealing with this sub-problem. We then solve equations (3–4) to (3–13) with binary variables x. The problem space is now smaller and we can get the final solution easier by using the MILP solver GUROBI.

The computational results obtained with NOSO and PASO are displayed in Table 4 and 5 and compared with the exact optimal solutions. Both heuristics are shown to produce approximate solutions close to optimal. In terms of computation time, PASO appears to be significantly less time consuming than NOSO.

6. Minimum Spanning Tree based Heuristic Algorithm (MSTH)

The idea of the Minimum Spanning Tree based Heuristic Algorithm is to avoid using too much resources on some nodes and links, especially in case of small capacity. The aim is to limit the possibility of using unnecessary bottleneck links and nodes, thus improving network utilization according to our problem objective.

6.1. Route Selection

In this section, we consider the problem of selecting the route that carries the traffic for each demand. Assume that we need to select a route and provision resources for a new demand $d \in D^{(t)}$ between source node s and destination node t. The main intuition behind the route selection algorithm is to find a route that (a) has enough capacity to host VNFs (Section 6.2) and (b) keeps the more critical links available for future demands.

The problem now can be stated as finding an optimal Max-Min path from s to t where $\underline{\sigma}(\mu) = Min\{w_{s,i_1}, w_{i_1,i_2}, .., w_{i_k,t}\}$ is maximized for each demand. This problem is recognized as the Maximum Capacity Path Problem [23] and is efficiently solved using a variant of Dijkstra's algorithm.

However, the above basic Maximum Capacity Path Problem only focuses on the utilization on paths or links (condition (b)). It is easy to realize that by using $\underline{\sigma}(\mu)$, the obtained routing path can violate condition (a) if we do not take into account the processing capacities on nodes. So, we propose to use the

following costs in extended Maximum Capacity Path Problem:

$$a_{uv} = w_{uv} * \frac{c_v}{\mathcal{C}} \quad (6\text{–}1)$$

where $\mathcal{C} = \max_{u \in V^{(t)}} c_u$

In summary, in the route selection section, we solve the extended Maximum Capacity Path Problem by using a modified Dijkstra algorithm to find a shortest path from s_d to t_d for each demand d in the network where each link is assigned a link weight a_{uv}, see detail in [23].

6.2. VNFs Distribution

After obtaining a set of paths $\Pi = \{P_1, P_2, ...P_{|D^{(t)}|}\}$ to route for demands, the last important step is how to distribute VNFs on those to minimize the node utilization. In this section, we will present a procedure to distribute efficiently VNFs on given paths. The principle is to put VNFs one by one on a path and consider the residual capacity of nodes on this path. The ideal distribution is that all needed resources for VNFs are shared equally for nodes on all paths. That means the ideal node utilization is computed by:

$$\mathcal{N}_{ideal} = \frac{\sum_{d \in D^{(t)}} \sum_{f \in F^d} q_f}{\sum_{i=1..|D^{(t)}|} \sum_{v \in P_i} c_v} \quad (6\text{–}2)$$

We consider the paths one by one. In particular, given demand d, we start from the first node on the path Π, and then put function $f_i : f_i \in F^d$ on node $v \in \Pi$ if

$$q_{f_i}/c_v \leq \mathcal{N}_{ideal} \quad (6\text{–}3)$$

and next to the succeeding node otherwise. Finally we locate the remaining functions on the destination of the path even if this is violated by equation (6–3).

Once the routes have been selected for each demand, it is possible that a given node (shared node) appears on r multiple paths. In this case, we would like to share fairly the capacity of this shared node on the r paths in order to avoid a poor utilization of the shared nodes. Therefore, we have:

- For each shared node v that appears at r paths, i.e. $P_{i_1}, ..P_{i_r}$, we set temporarily the available capacity of v on each path to $c'_v = c_v/r$.
- However, as we consider each path sequentially, it remains possible that the resource reserved on a shared node is not used in a given path if no VNF is assigned to this shared node. Therefore, whenever we finish distributing resources for a demand (in the list of demands), we have to check if the capacity of any shared node has not been used (satisfying the in-equation (6–3)) and then add its shared capacity fairly to this shared node on the remaining paths, to serve future demands. This step can improve significantly both the node utilization and the acceptation ratio of demands in our system.

Summary, the MSTH algorithm first finds the maximum capacity path with large node capacity for each demand. It then distributes efficiently VNFs on the resulting paths.

Table 1: Network parameter values for the case of unlimited resources and the case of limited resources.

Topology	Unlimited Resources		Limited Resources	
	W	C	W	C
Abilene	200	300	100	150
Nobel-eu	150	180	70	80
Nobel-germany	200	350	100	200
Test50	200	150	90	70
Test100	180	100	90	50
Test200	150	90	75	40

7. Performance Evaluation

In this section, we evaluate the performance of our proposed model via simulation. The experiments are executed in a Java based environment that allows running the simulation and optimization tool (Gurobi optimization [22]).

7.1. Simulation

In our simulations, we use real networks from SNDlib database [24]. Topologies (Test50, Test100, Test200) are generated by BRITE topology generation tool [25] using Waxman model with parameters $\alpha_w = 0.15, \beta_w = 0.2, m_w = 2$. The VNFs can be hosted at any node in the network. All nodes have the same capacity C. The capacity of the links was set to W. For each topology, we have considered two values for the pair (C, W), which we refer to as unlimited and limited resources, as listed in Table 1.

There are a set of $F = 10$ VNFs available, each VNF requires a computation capacity $q_f = \{1, 2, .., 10\}$. Demands require to run service chains composed of 5 VNFs from F. For each demand, its source and destination are generated uniformly at random from all

Table 2: The Solution Efficiency with and without Flow Covers for various values of τ in case of unlimited resources

Instances	$\|F\|$	$\|D^{(t)}\|$	n	τ	Obj Value	Link load ($\mathcal{L}$)	Node Load ($\mathcal{N}$)	$\|S0\|$	Number of generated flow covers	GAP	AcceptNo	Time of MILP solver
Abilene	5	100	12		8.88	0.33	0.7800	Without flow covers		0.00	100	16.47
				1	8.88	0.33	0.78	46	1953	0.00	100	12.85
				2	8.88	0.33	0.78	46	31032	0.00	100	13.32
				3	8.88	0.33	0.78	46	424358	0.00	100	24.01
				4	8.88	0.33	0.78	46	3831782	0.00	100	187.38
Nobel-eu	5	100	28		9.09	0.34	0.56	Without flow covers		0.00	100	4000.00
				1	9.09	0.34	0.56	30	1675436	0.00	100	4000.00
				2	9.09	0.34	0.56	30	6030415	0.00	100	4000.00
				3	9.08	0.34	0.57	30	24725614	0.00	100	4000.00
				4	9.06	0.34	0.58	30	20373253	0.00	100	4000.00
Nobel-germany	5	100	17		9.38	0.28	0.33	Without flow covers		0.00	100	181.84
				1	9.38	0.28	0.33	37	156	0.00	100	70.34
				2	9.38	0.28	0.33	38	795	0.00	100	76.44
				3	9.38	0.28	0.33	38	6285	0.00	100	95.20
				4	9.38	0.28	0.33	38	39099	0.00	100	147.05
Test50	5	100	50		9.15	0.57	0.27	Without flow covers		0.00	100	9.40
				1	9.15	0.57	0.27	32	509	0.00	100	8.65
				2	9.15	0.57	0.27	32	7379	0.00	100	10.91
				3	9.15	0.57	0.27	32	62335	0.00	100	16.30
				4	9.15	0.57	0.27	32	669444	0.00	100	51.79
Test100	5	100	100		8.75	0.96	0.28	Without flow covers		0.00	100	745.33
				1	8.75	0.96	0.28	25	1619	0.00	100	1133.28
				2	8.75	0.96	0.28	25	12376	0.00	100	1958.50
				3	8.74	0.96	0.29	25	352334	0.00	100	4000.00
				4	8.74	0.96	0.29	25	2869554	0.00	100	4000.00
Test200	5	100	200		9.11	0.56	0.32	Without flow covers		0.00	100	47.16
				1	9.11	0.56	0.32	21	2709	0.00	100	42.74
				2	9.11	0.56	0.32	21	37373	0.00	100	65.54
				3	9.11	0.56	0.32	21	462335	0.00	100	71.95
				4	9.11	0.56	0.32	21	5669438	0.00	100	48.92

possible (s^d, t^d) pairs. Demand capacity requirements are selected from a set $b^d = \{1, 2, .., 10\}$ (bandwidth unit).

We run a first experiment to evaluate the solution efficiency of our proposed model and identify the size of the problems that can be solved exactly. We consider a set of demands $D^{(t)} = 100$ and different topologies. We also evaluate the improvement in efficiency when generating flow cover inequalities as compared to the case without flow covers.

In a second experiment, we assess various metrics in a dynamic system such as the blocking rate, the resource utilization and the average length of demand's path. In our simulation, demand arrivals are generated using a Poisson process of parameter λ chosen such that the overall blocking is below 0.2. The holding time of each demand is generated from an exponential process with an average parameter 3. We generate 10 runs of length T = 2000 seconds each. The results are averaged out over the runs and we compute the 95% confidence interval for the results using a small-sample statistic.

The simulation keeps the current state of the network as a list of ongoing demands including the location of functions and their routing path. We update the system after a certain time depending on the system configuration. In our simulation, we update the system after $\Delta t = 3ms$. Every Δt, we first check if any demand had terminated during the previous inter-arrival time, remove them from the list and then release the resources that they were holding. Then, we run both the MILP algorithm and the heuristics to find the solution. Depending on the algorithms under evaluation, accepted demands are added to the list of ongoing demands and the available resources are updated simultaneously.

In the next section, we evaluate the efficiency of our proposed algorithms in terms of input size and network performance.

Table 3: The Solution Efficiency with and without Flow Covers for various values of τ in case of limited resources

Instances	$\|F\|$	$\|D^{(t)}\|$	n	τ	Obj Value	Link load ($\mathcal{L}$)	Node Load ($\mathcal{N}$)	$\|S0\|$	Number of generated flow covers	GAP	AcceptNo	Time of MILP solver
Abilene	5	100	12		8.18	0.92	0.89	Without flow covers		0.00	100	12.15
				1	8.18	0.92	0.89	14	6	0.00	100	14.12
				2	8.18	0.92	0.89	14	13	0.00	100	12.97
				3	8.18	0.92	0.89	14	43	0.00	100	15.17
				4	8.18	0.92	0.89	14	113	0.00	100	13.69
Nobel-eu	5	100	28		8.30	0.80	0.90	Without flow covers		0.00	100	627.25
				1	8.30	0.80	0.90	12	2343	0.00	100	245.75
				2	8.30	0.80	0.90	12	10340	0.00	100	350.36
				3	8.30	0.80	0.90	12	29849	0.00	100	337.34
				4	8.30	0.80	0.90	12	69312	0.00	100	375.48
Nobel-germany	5	100	17		8.48	0.72	0.79	Without flow covers		0.00	100	140.57
				1	8.48	0.72	0.79	19	310	0.00	100	119.28
				2	8.48	0.72	0.79	20	1658	0.00	100	126.89
				3	8.48	0.72	0.79	20	7341	0.00	100	89.84
				4	8.48	0.72	0.79	20	34128	0.00	100	69.78
Test50	5	100	50		6.37	0.52	1.00	Without flow covers		0.02	79	4000.00
				1	6.37	0.52	1.00	14	974137	0.02	79	4000.00
				2	6.37	0.52	1.00	14	6968816	0.02	79	4000.00
				3	6.35	0.54	1.00	14	28845999	0.03	79	4000.00
				4	6.37	0.52	1.00	14	70807063	0.02	79	4000.00
Test100	5	100	100		7.27	0.98	0.54	Without flow covers		0.00	88	4000.00
				1	7.27	0.98	0.54	11	879	0.00	88	4000.00
				2	7.27	0.98	0.54	11	4964	0.00	88	4000.00
				3	7.27	0.98	0.54	11	123489	0.00	88	4000.00
				4	7.23	0.98	0.58	11	868324	0.00	88	4000.00
Test200	5	100	200		7.87	1.00	0.62	Without flow covers		0.00	95	4000.00
				1	7.87	1.00	0.62	12	1257	0.00	95	4000.00
				2	7.87	1.00	0.62	12	3806	0.00	95	4000.00
				3	7.87	1.00	0.62	12	147960	0.00	95	4000.00
				4	7.87	1.00	0.62	12	3580647	0.00	95	4000.00

For all the tables, "$|F|$" is the number of functions and "$|D^{(t)}|$" is the number of arrived demands at time (t), the "Obj Value" is the value of the objective function computed by equation (3–1), where the "Link Load" and the "Node Load" are defined by equation (3–2), (3–3) respectively. Moreover, "$|S_0|$" is the size of cover sets, "GAP" is obtained directly from Gurobi optimization tool, "AcceptNo" is the number of accepted demands.

7.2. The Solution Efficiency with and without Flow Covers for various values of the parameter τ.

Tables 2 and 3 show the improvement in efficiency by generating flow covers for solving MILP with different values of $\tau = \{1, 2, .., 5\}$ (τ is defined earlier in Section 4). We run the MILP solver, GUROBI [22], using the default parameter values and generating flow covers as described in Section 4 to evaluate the efficiency of our method using Flow Covers.

We observe that generating flow covers with small value of τ makes the solver more efficient and the optimal solution is obtained much faster (see the value of objective and GAP of the MILP solver). For instance, using the Abilene topology, we can get the optimal solution after 12.8553 seconds when generating some flow covers, whereas we need 16.4741 seconds without adding flow covers.

A second observation is the impact of the parameter τ in the computation time of the MILP solver. As explained in Section 4, τ is used to increase the opportunity of finding good flow cover cuts. The idea is to find stronger flow cover inequalities for the MILP solver. However, as we can see in Tables 2 and 3, larger values of τ do not bring any benefit in the computation time of the MILP solver. This is because when increasing the value of τ, it takes more time to find $C^k_{k+\tau}$ cover sets for each link $(u, v) \in E^{(t)}$ where $k <= |D^{(t)}|$. As a consequence, these extended flow covers are not strong for the problem. In addi-

Table 4: Comparison of the objective values obtained with the various proposed algorithms in case of unlimited resources

Instances	F	D	n	Optimal Solution			MSTH			NOSO			PASO with adding Flow Cover		
				Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time
Abilene	5	100	12	8.88	100	1345.60	8.80	100	0.15	8.88	100	80.57	8.88	100	8.64
Nobel-eu	5	100	28	9.09	100	4008.06	8.93	100	0.3	9.08	100	4013.16	9.08	100	1666.55
Nobel-germany	5	100	17	9.38	100	4005.29	9.29	100	0.26	9.37	100	1149.81	9.38	100	45.75
Test100	5	100	100	8.75	100	4031.37	8.55	100	0.23	8.75	100	211.78	8.75	100	130.79
Test200	5	100	200	9.11	100	4193.39	8.95	100	0.34	9.11	100	309.45	9.11	100	165.93
Test50	5	100	50	9.15	100	4011.21	9.08	100	0.33	9.15	100	27.79	9.15	100	17.93

Table 5: Comparison of the objective values obtained with the various proposed algorithms in case of limited resources

Instances	F	D	n	Optimal Solution			MSTH			NOSO			PASO with adding Flow Cover		
				Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time	Obj Value	AcceptNo	CPU Time
Abilene	5	100	12	8.18	100	4003.19	8.02	100	0.33	8.18	100	235.38	8.18	100	6.33
Nobel-eu	5	100	28	8.30	100	4011.39	7.96	99	0.27	8.27	100	4062.13	8.30	100	135.71
Nobel-germany	5	100	17	8.48	100	4005.79	8.35	100	0.42	8.48	100	4004.94	8.48	100	81.69
Test100	5	100	100	7.27	88	4030.55	5.02	69	0.23	7.27	88	5042.80	7.27	88	3664.51
Test200	5	100	200	7.87	95	4214.60	5.13	71	0.43	7.87	95	4583.08	7.87	95	3296.22
Test50	5	100	50	6.37	79	4012.06	5.77	78	0.35	6.37	79	4614.23	6.37	79	3720.65

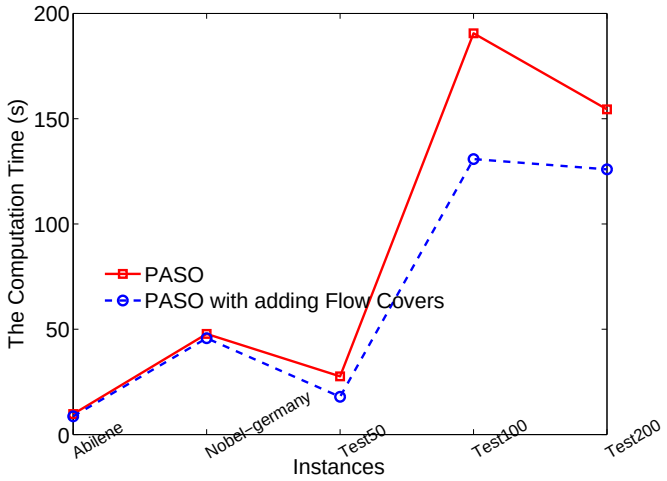


Figure 2: The average running time per demand vs λ

tion, a few additional computation time is required to find cover sets with large value of τ.

Clearly, the more flow covers are generated, the stronger the linear relaxation. However generating flow covers requires significant time to find cover sets that relates directly to the parameters k and τ. Note that the value of k is computed in Section 4 depending on the system parameters such as the link capacity $w_u v$ and the bandwidth requirement of demands b^d. Determining good values of τ can only be done on an experimental basis. From our experiments, it is observed that the best value of τ in our simulation is 1.

7.3. Comparison of the Objective Value of the Various Proposed Heuristics

Tables 4 and 5 compare the objective value of the proposed heuristic algorithms. It is observed that our model can find the optimal solution for a large instance, namely, up to 200 nodes and 100 demands. The objective value obtained by both NOSO and PASO is close to the optimal. However, PASO is observed to be significantly more efficient than NOSO in terms of computation time. Regarding MSTH, the objective value is quite close to the optimal solution in case of unlimited resources, see Table 4. However the difference with the exact optimal objective values significantly increases in case of limited resources, see Table 5. Therefore, our MSTH heuristic turns out to be accurate in the case of unlimited resources only. In term of the computation time, we can realize that the MSTH always got the highest computation efficiency comparing with NOSO and PASO. For instance, it can find a solution 57.33 times faster than PASO, 537.13 times faster than NOSO and 9566 times faster than the exact approach with the unlimited resource Abilene topology. This clearly shows the wide range of trade-offs between optimality and computational efficiency that can be obtained through our various heuristics as well as our exact approach for the problem.

7.4. The improvement in the Computation Time of using Flow Covers in PASO Heuristic Algorithm

Fig. 2 illustrates the computation time of PASO algorithm in two cases: with adding Flow Covers and

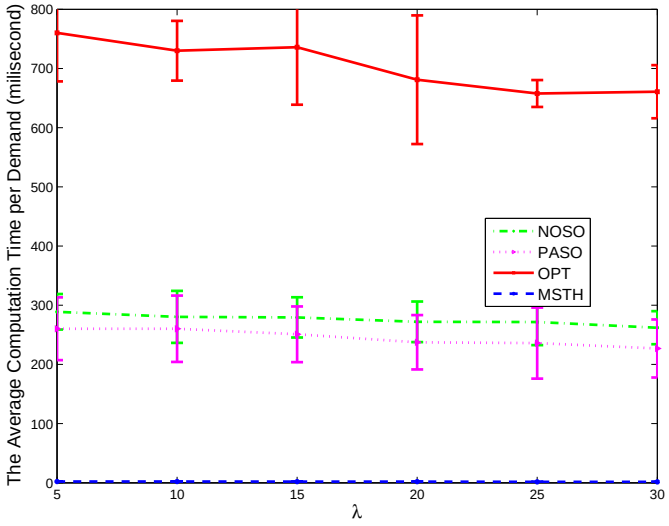


Figure 3: The average running time per demand vs λ

without adding Flow Covers. Obviously, adding Flow Covers improves significantly the computation time for the PASO algorithm, especially in a large network, such as saving about $50 - 100$ seconds in *Test 100*, *Test 200*. It again illustrates the efficiency of our method for generating Flow Covers to solve the MILP problem.

7.5. The Impact of The Arrival Rate of Demands

This section evaluates the network metrics in the dynamic scenarios. We use the largest topology (*Test200*) and demand arrives as a Poisson process with $\lambda = 5..30$.

Fig. 3 depicts the average computation time per demand for our three algorithms. In terms of execution time, the MILP algorithm requires about 800 seconds per demand, NOSO and PASO requires about 300 seconds, whereas MSTH requires only 0.02 seconds.

Especially, considering in detail the computation time in Tables 4 and 5, PASO (the dotted line or pink line) run significantly faster (from 2 to 20 times) than NOSO (the dash-dot line or green line). For example, with Abilene instance, PASO can obtain the solution after 8.641 seconds while NOSO needs 80.579 seconds. As described earlier in Section 5, the idea of NOSO is to find the best allocations for VNFs on network nodes before routing them satisfying the order of VNFs where as PASO focuses on finding the shortest path between the source and the destination of demands before locating VNFs on obtained paths. The results of Fig. 3 show that considering the *PATH* optimization before the *NODE*-locating optimization

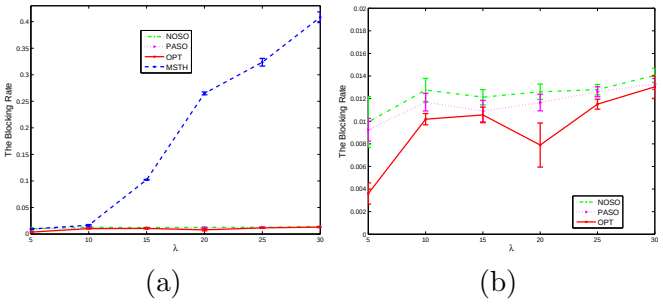


Figure 4: The blocking rate vs λ

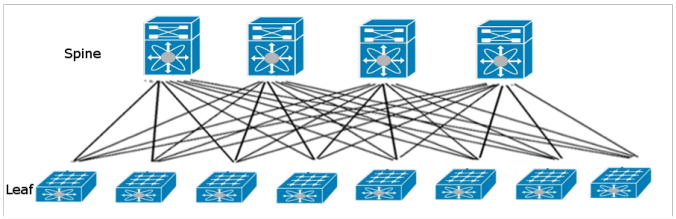


Figure 5: Typical Spine-and-Leaf Topology

makes the problem easier to solve. That is interesting and important observation because, in the literature, all previous heuristics solved the VNF-locating phase before the routing phase to find the feasible solution.

Fig. 4a shows the blocking rate of the topology Test200, for three heuristics as well as the optimal solution obtained with the MILP algorithm. We observe that the MSTH algorithm diverges very quickly when λ increases. However, figure 4(b) confirms that the MILP algorithm (OPT) provides the optimal solution with the smallest blocking rate and that our NOSO and PASO are close to the optimal.

Moreover, regarding the maximum link utilization (Fig.7) and the average path length (Fig. 8), both PASO and NOSO yield solutions very close to the exact optimal MILP solution.

As a consequence, the PASO algorithm appears as the best option, however, the MSTH algorithm provide us with a feasible solution in a very limited time and could be considered as a candidate when a solution need to be computed very quickly in a large network.

7.6. The Impact of Resource Capacity on Data Centre Topology

In this section, we investigate the impact of the resource capacity on Data Centre topology with our proposed heuristics. We use the Leaf-Spine topology that is a novel architecture used in virtualized

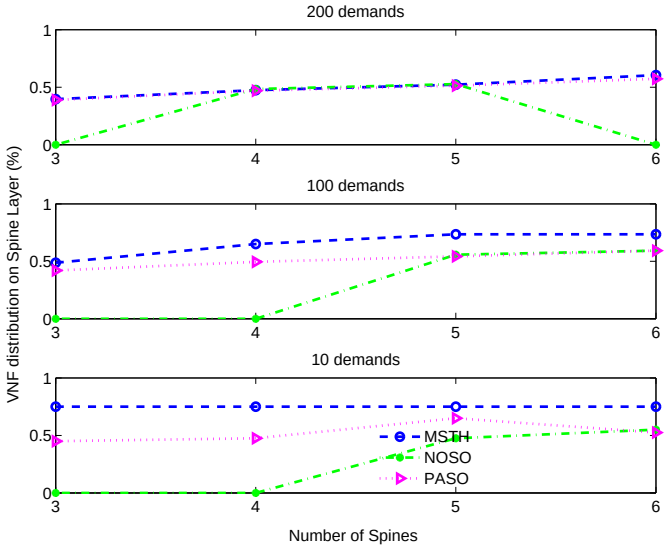


Figure 6: The VNFs distribution on a Leaf-Spine topology

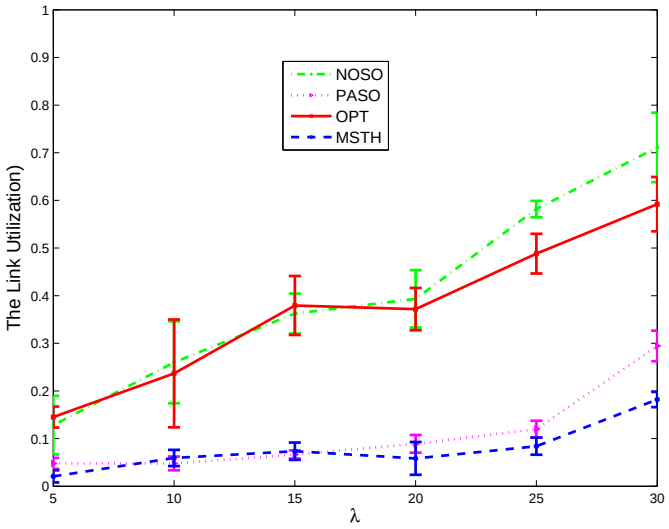


Figure 7: The maximum link load vs λ

technologies (Fig. 5, [26]). Unlike traditional core-aggregation-access layer network, the leaf-spine architecture has only two layers. These leaf switches are fully meshed to a series of spine switches. The advantage of this topology is the ease of adding hardware and capacity. Moreover, latency improves and bottlenecks are minimized. To the best of our knowledge, this is the first attempt at exploring VNFs distribution and performance evaluation in this Data Centre topology. In our simulation, we set the capacity of the spine switches four times larger than those of leaf switches.

First, we estimate the distribution of VNFs onto the Spine Layer for this topology for our three proposed heuristics (PASO, NOSO, and MSTH). Fig.6

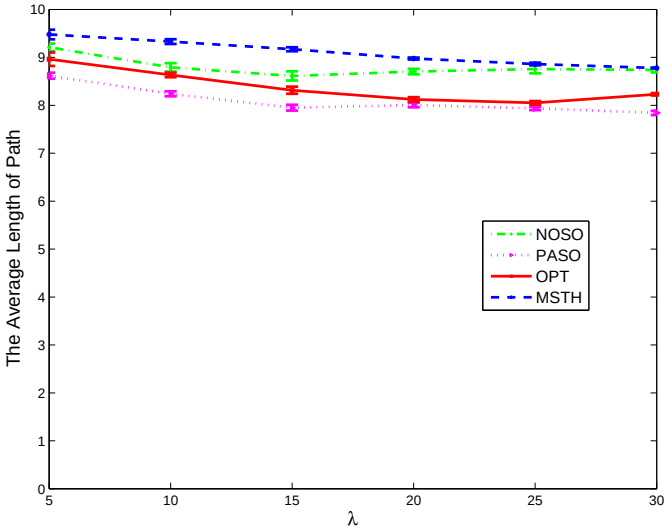
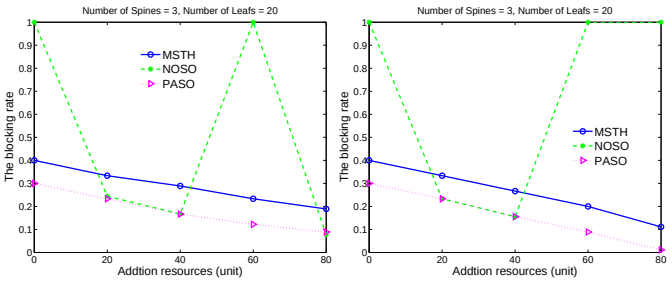


Figure 8: The average length vs λ

shows the VNFs percentage on the Spine Layer for four instances of the Leaf-Spine topology with different number of demands (10, 100 and 200 demands). The number of Spines in these instances increase from 3 to 6, while we keep the number of Leafs constant (equals 20). Our first observation is that the NOSO algorithm performs poorly. Indeed, with NOSO heuristic, VNF distribution equals 0 when it can not find any solution. This is due to the fact that with NOSO, we solve the sub-problem to find the location of VNFs before finding the routing path for each demand. Therefore, there exist some cases where we can not find any path satisfying the resource constraints. In addition, the figures show that for all our heuristics, increasing the number of Spines does not impact significantly the VNF distribution on this topology. In general, the MSTH algorithm distributes VNFs almost on Spine Layer (75%), while the NOSO and PASO algorithms distribute VNFs equally between the Spine Layer and the Leaf Layer. It is a trend that VNFs should be fairly distributed on both layers to avoid the congestion on any layer.

In the second experiment, we show the impact of adding resource capacity on this topology. We use a Leaf-Spine topology with a number of Spines = 3 and number of Leafs = 20 with a total capacity = 192($units$). We consider two assumptions. First, we keep the capacity of every node on the Spine Layer and then add more node capacity on the Leaf Layer using a random distribution. Fig.9(a) shows the blocking rate for our three heuristics. We observe again the poor performance of the NOSO algorithm that can not find any solution in some instances (in those



(a) Adding to Leaf Layer (b) Adding to Spine Layer

Figure 9: The blocking rate vs Resource capacity on Data Centre Topology

cases the blocking rate equals 1). This is because we solve the sub-problem of locating VNFs before finding the routing path for each demand. Adding the Leaf switches provide more opportunities to put VNFs. However, there is no connectivity between Leaf switches in the Leaf-Spine topology, therefore, no path is found to route the demands through the required VNFs hosted on those nodes. Obviously, the blocking rate can be reduced by adding capacity on the Leaf Layer. Alike for the first experiment, the PASO heuristic obtains the best blocking rate. Second, we implement a scenario where we keep the capacity of every node on the Leaf Layer constant, and increase the capacity of nodes on the Spine Layer, see Fig.9(b). The figure shows that the PASO heuristic always performs best in all situations. Obviously, adding hardware and capacity on both layers of the Leaf-Spine topology increases the network performance (we cut the blocking rate by 10% with an increased of 10% of the capacity).

Finally, to which layer should we add capacity in order to get the highest benefit? Fig. 10 shows the performance improvement of PASO when we add resources on Spine Layer and Leaf Layer. As we can see in the figure, adding resources to Spine Layer obtains slightly more profit than to Leaf Layer. This is because we have more spaces on the centre and therefore we can reduce the congestion for the users. However, it also increases the usage of the back-haul links of the network.

8. Conclusions and Future Work

In this paper, we consider an NFV dynamic system, study the optimization of its resources and provide guidelines for its design. We propose a new efficient Mixed Integer Linear Programming model while taking into account the decision of Service Providers

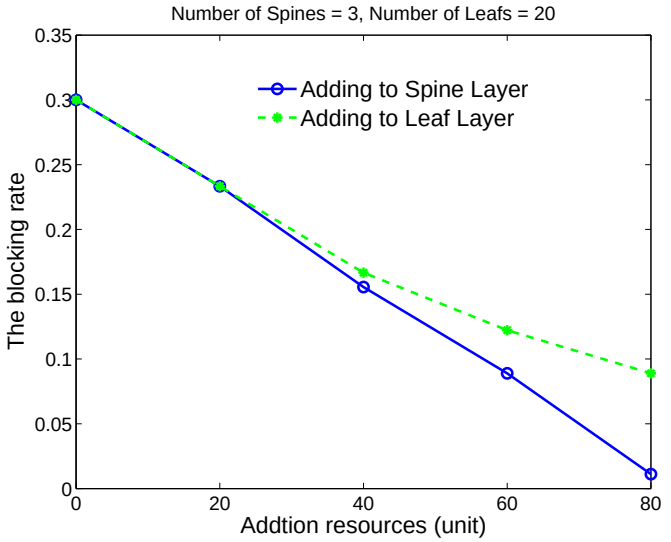


Figure 10: Performance improvement of PASO when adding resources

to serve or reject a demand. The overall solution is enhanced by generating some strong flow covers for the MILP solver. The main contribution, in contrast with existing work, is that the model is linear and can find the optimal solution for large network instances, typically for systems as large as 200 nodes and 100 demands. Furthermore, we propose three efficient heuristics to find good quality feasible solutions for larger instances with a reduced execution time. To complete our study, we evaluate our solution in various scenarios and use an event-driven simulation to assess some realistic metrics in an NFV dynamic system. We provide several valuable observations and guideline for the efficient design of such systems. Future work plans to evaluate our heuristics using testing facilities where we can deploy our solution on a real environment.

References

[1] J. G. Herrera, J. F. Botero, Resource allocation in NFV: A comprehensive survey, IEEE Transaction Network Service Management 13 (3) (2016) 518–532. doi:10.1109/TNSM.2016.2598420.

[2] X. Li, C. Qian, A survey of network function placement, in: 13th IEEE Annu. Consumer Communication & Network Conference, CCNC 2016, January 9-12, pp. 948–953.

[3] A. Leivadeas, M. Falkner, I. Lambadaris, G. Kesidis, Dynamic traffic steering of multi-tenant virtualized network functions in SDN enabled data centers, in: 21st IEEE International Workshop on Computer Aided Modelling and Design of Communication Links and Networks, Toronto, ON, Canada, 2016, pp. 65–70.

[4] C. A. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, Á. Monje, On the op-

timal allocation of virtual resources in cloud computing networks, IEEE Trans. Computers 62 (6) (2013) 1060–1071.

[5] T. Lin, Z. Zhou, M. Tornatore, B. Mukherjee, Demand-aware network function placement, Journal Lightwave Technology 34 (11) (2016) 2590–2600.

[6] T. Lukovszki, S. Schmid, Online admission control and embedding of service chains, in: Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO, Montserrat, Spain, 2015, pp. 104–118.

[7] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. Wilfong, Y. Yang, C. Guo, PACE: policy-aware application cloud embedding, in: Proc. IEEE INFOCOM 2013, Apr. 14-19, pp. 638–646.

[8] J. Elias, F. Martignon, S. Paris, J. Wang, Efficient orchestration mechanisms for congestion mitigation in nfv: Models and algorithms, IEEE Transactions on Services Computing (2015) 534–546.

[9] M. Obadia, J. L. Rougier, L. Iannone, V. Conan, M. Bouet, Revisiting NFV orchestration with routing games, in: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, November 7-10, 2016, 2016, pp. 107–113.

[10] T.-M. Nguyen, S. Fdida, T.-M. Pham, A Comprehensive Resource Management and Placement for Network Function Virtualization, in: The 3rd IEEE Conference on Network Softwarization (IEEE NetSoft 2017), Bologna, Italy, 2017.

[11] Q. Sun, P. Lu, W. Lu, Z. Zhu, Forecast-assisted NFV service chain deployment based on affiliation-aware vnf placement, in: 2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016, 2016, pp. 1–6.

[12] W. Ma, C. Medina, D. Pan, Traffic-aware placement of NFV middleboxes, in: 2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015, 2015, pp. 1–6.

[13] J. Liu, W. Lu, F. Zhou, P. Lu, Z. Zhu, On dynamic service function chain deployment and readjustment, Vol. 14, 2017, pp. 543–553.

[14] B. Addis, D. Belabed, M. Bouet, S. Secci, Virtual network functions placement and routing optimization, in: 4th IEEE International Conference on Cloud Networking, CloudNet 2015, Niagara Falls, ON, Canada, October 5-7, 2015, 2015, pp. 171–177.

[15] T. Kuo, B. Liou, K. C. Lin, M. Tsai, Deploying chains of virtual network functions: On the relation between link and server usage, in: 35th Annual IEEE International Conference on Computer Communications, INFOCOM, San Francisco, CA, USA, 2016, pp. 1–9.

[16] H. Yang, J. Zhang, Y. Ji, R. Tian, J. Han, Y. Lee, Performance evaluation of multi-stratum resources integration based on network function virtualization in software defined elastic data center optical interconnect, Vol. 23, OSA, 2015, pp. 31192–31205.

[17] H. Yang, J. Zhang, Y. Ji, Y. He, Y. Lee, Experimental demonstration of multi-dimensional resources integration for service provisioning in cloud radio over fiber network, in: Scientific Reports, 2016/07/28/online.

[18] M. W. Padberg, T. J. V. Roy, L. A. Wolsey, Valid linear inequalities for fixed charge problems, Operations Research 33 (4) (1985) 842–861.

[19] T. J. V. Roy, L. A. Wolsey, Valid inequalities for mixed 0-1 programs, Discrete Applied Mathematics 14 (2) (1986) 199–213.

[20] T. J. Roy, L. A. Wolsey, Solving mixed integer programming problems using automatic reformulation, Operations Research 35 (1) (1987) 45–57.

[21] Z. Gu, G. L. Nemhauser, M. W. P. Savelsbergh, Lifted cover inequalities for 0-1 integer programs: Computation, INFORMS Journal on Computing 10 (4) (1998) 427–437.

[22] [link].
URL `www.gurobi.com`

[23] M. Gondran, M. Minoux, Graphs, Dioids and Semirings. Springer, 2008.

[24] S. Orlowski, R. Wessäly, M. Pióro, A. Tomaszewski, Sndlib 1.0 - survivable network design library, Networks 55 (3) (2010) 276–286.

[25] A. Medina, A. Lakhina, I. Matta, J. W. Byers, BRITE: an approach to universal topology generation, in: 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001), 15-18 August 2001, Cincinnati, OH, USA, 2001, p. 346.

[26] Cisco data center spine-and-leaf architecture: Design overview white paper, April 15, 2016 (April 15, 2016).
URL `https://www.cisco.com/`