# Online and batch algorithms for VNFs placement and chaining

Oussama Soualah, Marouen Mechtri, Chaima Ghribi, Djamal Zeghlache

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128618302548
Manuscript_8039c9f716e1c7e8efcb7e9db80de09a

# Online and Batch Algorithms for VNFs Placement and Chaining

Oussama Soualah[a], Marouen Mechtri[b], Chaima Ghribi[a], Djamal Zeghlache[a]

[a]*Mines-Telecom, Telecom SudParis, CNRS UMR 5157, Evry, France*
*{oussama.soualah, chaima.ghribi, djamal.zeghlache}@telecom-sudparis.eu*
[b]*Orange Labs, Chatillon, France*
*marouane.mechteri@orange.com*

**Abstract**

This paper proposes an Integer Linear Program (ILP) to address the Virtualized Network Function Forwarding Graph (VNF-FG) placement and chaining problem when VNFs are shared across tenants to optimize resource usage and increase provider revenue. Since ILP based approaches do not scale well with problem size, the proposed algorithm (R-ILP for reduced exploration) selects a limited number of candidate hosts from the infrastructure to control complexity. Since the online R-ILP treats the requests sequentially, a batch strategy that operates on a set of requests is also proposed to improve performance. The online algorithm processes the VNF-FG requests on a sequential basis as they arrive while the batch mode treats several requests jointly over a batch window. This work focuses on energy consumption optimization as a general objective. The proposed solutions are shown to outperform competitor algorithms from the state of the art that rely also on VNFs sharing. Results from extensive simulations, based on realistic and large scale topologies, report the performance in terms of rejection of service requests, energy consumption, scalability and achieved revenues. The performance benefits of operating our R-ILP in batch mode are highlighted.

*Keywords:* NFV, VNF-FG Placement and Chaining, Integer Linear Program, Batch, Energy efficiency, VNF sharing

## 1. Introduction

For Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) to capture the market, network services vendors and providers must overcome a number of challenges in service modeling and orchestration, in optimization of their shared and virtualized resources and to take advantage of open source technologies. Network and service providers need also to develop the ability to provide their cloud and network services on demand at minimum infrastructure cost while maximizing their revenues. These revenues can be achieved if the providers maximize virtualized functions and physical resources usage and automate the provisioning and management of the services to multiple tenants. By optimizing the placement and chaining of virtualized network functions, the providers can truly cash on the benefits of NFV and SDN.

The work presented here is an extension of our previous work [1], with the following main added values: i) a pragmatic batch algorithm, ii) a comparison with related work strategies, iii) an in depth performance evaluation as well as iv) a new selection method to keep prominent candidates. It focuses on supporting the cloud and network services providers in this required overall optimization. To address their needs, the sharing of Virtualized Network Functions (VNFs) across multiple tenants and the optimized placement and chaining of these functions are taken into account in our contribution. VNFs may be (time) shared to avoid spawning more VNFs to reduce hosting cost and power consumption. Placing a sequence of VNFs providing network services (e.g., load balancers, firewalls, IPS/IDS, etc.) while steering traffic flows across the VNFs according to a VNF-FG specification is an NP-Hard problem [2].

Our contribution consists in making available an Integer Linear Program (ILP) to improve the placement and chaining of NFV services to meet the aforementioned objectives and the need for algorithms that are sufficiently generic to be tuned according to the preferences and business interests of the providers. In modeling the placement of VNF requests, this need is integrated in our ILP with criteria and constraints that can be configured as desired. Another important facet of overall cost optimization is the energy consumption induced by the algorithms that should also be minimized. The proposed ILP includes energy efficiency in the optimization objective. Other objectives based on the provider needs, such as minimization of requests rejection and maximization of the earned revenue, can be straightforwardly added to the algorithm.

As mentioned, the objective of this paper is to provide a generic ILP that addresses the online VNF-FG placement and chaining problem using the sharing capability enabled by the NFV paradigm and that can be tuned to meet the requirements and preferences of the stakeholders. A batch mode is also proposed to treat jointly a group of requests over a time window to improve overall performance and optimization compared with the sequentially operated online ILP. In both modes we select a subset from the most prominent candidate hosts to improve scalability with marginal impact on optimality.

The performance of the proposed algorithms is compared with current state of the art algorithms known to perform best among the algorithms available in the literature that address both placement and chaining of shared VNFs. The first algorithm we compare performance with is the ILP based algorithm of [3] that considers the objective of minimizing energy consumption for VNF placement and chaining. The proposed robust solution in [3] considers a joint resources and flow routing assignment problem for VNFs placement with the objective of minimizing power consumption of servers and switches and the routing graph. The second is SFC-Monte Carlo Tree Search (MCTS for short), a tree search based algorithm that also shares VNFs across users or tenants to improve resource usage and minimize cost [4]. We limit the comparison to these two algorithms since they are, to the best of our knowledge, the closest to our work. MCTS with and without sharing of VNFs[4] has also be shown to perform among the best VNF placement and chaining algorithms [4] [5] in the current state of the art as described in the related work section of this document. Results of extensive simulations show that our proposed R-ILP outperforms MCTS in all reported metrics and performance indicators. The ILP also scales as shown in the complexity mathematical expression, reported in section 4.2, since only a limited number of candidate hosts are explored for placement and chaining and this is shown not to degrade performance of the ILP, an added advantage compared to MCTS and the state of the art. These results also confirm that it is possible to use an ILP based algorithm, with the appropriate depth of exploration of the solutions, in our case by retaining only a limited number of candidate hosts for placement and chaining, to avoid resorting to heuristics for large problem sizes while finding good placement solutions. Besides, our batch approach, that makes use of our algorithm R-ILP, outperforms the other on line strategies. This result highlights the efficiency of the batch treatment and emphasizes the need to investigate more this mode that serves more clients on more provider resources thus improving scalability with problem size.

Section 2 of this paper presents related work addressing VNF-FG placement and chaining. The system model and the formal description of the problem statement are described in Section 3. Section 4 presents the proposed ILP model, the online and the batch algorithms. Section 5 reports the performance evaluation results that confirm and highlight the expected improvements. Section 6 summarizes the main findings.

## 2. Related work

The VNF placement and chaining problem has received considerable attention in recent years. In fact, it was overviewed for instance in [6]. Prior works typically cast the problem into a traditional virtual network embedding (VNE) problem. However, the placement and routing of VNFs is a problem fundamentally different from the VNE problem [7], [8], [9]. In VNE, requests are modeled by simple undirected graphs, whereas VNF chains are more complex components that contain both the VNFs to place and the traffic flows to steer between the end points. In addition to that and contrary to the VNE problem, the VNF placement and routing demand is not a multipoint-to-multipoint network connection request, but a point-to-point source-destination flow routing demand.

The proposed solutions for the VNF placement and chaining can be classified in heuristic and exact approaches. Exact methods are efficient in finding optimal solutions only for small problem sizes and are not usually suitable for data centers with thousands of physical machines. To improve the scalability problem of exact methods, heuristics are used to handle larger infrastructures. The heuristics typically reduce execution times and complexity at the expense of optimality. We review the related work by organizing the state of the art into these two main classes.

### *2.1. Heuristic approaches*

The work in [10] proposes a heuristic algorithm to place VNFs with a minimum overall network cost objective. Authors in [11] provide a "network functions virtualization " orchestration model and design a greedy heuristic to achieve the placement. A heuristic proposed in [12] uses Simulated Annealing (SA) to find solutions in shorter times but simplifies the overall problem by considering only one type of VNF and rather small chains. In the above mentioned solutions, the problem of VNF placement and chaining is solved in two steps. The VNFs are placed first and then the traffic is steered through the chains via the shortest possible paths but this leads to suboptimal solutions.

In [13], authors present an energy aware Game Theory approach to resource allocation for VNFs in NFV environments but evaluate their solution on a small infrastructure involving just 3 and 5 servers. Related work presented in [2] proposes a polynomial complex-

ity heuristic for VNF placement and chaining using a complex transformation of the problem by adding new virtual switches but this increases complexity. The proposed heuristic models the problem as a Multi-Stage directed graph with associated costs. The VNF are placed using the Viterbi algorithm [14] applied to the Multi-Stage graph. The proposed Multi-Stage algorithm has the drawback of favoring the mapping of nodes (VNFs) of each request on the same physical server. The bandwidth constraints between interconnected VNFs are hence implicitly neglected each time VNFs are embedded in the same physical machine.

In [15], authors proposed an eigendecomposition based approach for joint VNFs placement and traffic steering in forwarding graphs. A greedy algorithm based heuristic is also presented to solve the problem iteratively. The Greedy solution is based on bipartite graph construction and matching techniques and solves the problem in two steps (mapping VNFs then steering traffic between them).

In [16], authors address the problem of VNF Placement and traffic steering in Cloud datacenters and design a dynamic programming (DP) algorithm for VNF chain placement that runs in polynomial time. Authors highlight the efficiency of a dynamic programming approach to solve the VNF embedding problem.

In [5] authors proposed a decision tree based algorithm to address the problem of VNFs placement and chaining. This work makes use of the Monte-Carlo Tree Search algorithm to control the problem complexity. Simulation results show that this new algorithm outperforms the state of the art approaches especially in rejection rate of placement requests. The main disadvantage of this approach is that it does not handle graph based requests and only deals with chains. Authors of [17] extended this work by providing a reliable algorithm that ensures resilience to physical link failures. This was one of the first papers , as well as [18], addressing reliability and survivability for VNFs placement and chaining problem.

Authors in [19] propose an online algorithm for availability-aware SFC mapping for wide area service chaining, which can minimize the resources allocated to service chain requests while meeting clients' availability requirement. Authors in [20] propose a matrix-based method and a multi-stage algorithm for the placement of service chains. These solutions support only the online mode.

In [4], authors propose a VNF sharing based algorithm that combines the advantages of the Monte-Carlo Tree Search algorithm and the good performance achieved by the strategy presented in [5]. Sharing of the VNFs is taken into account in this new algorithm to use more efficiently the physical resources. This algorithm is shown to outperform most other state of the art algorithms in energy consumption and rejection rate of placement requests.

The major limitation of this kind of heuristic based algorithms is the absence of guarantee of an optimal solution.

### *2.2. Exact approaches*

Authors of [3] propose a Mixed Integer Linear Program (MILP) for VNFs placement and use the Cplex solver to find solutions that minimize energy consumption. The Author of [21] provide a proof that MILP problems are NP-Hard. The MILP solution of [3] is for these reasons evaluated for only a small topology including just 12 physical machines and 16 service chains. In [12],[22], [23], [24], [2] and [8] VNF placement and chaining are also formulated and solved as a MILP. A MILP was also proposed in [25] to formulate service placement and flow routing, and also to minimize the resource utilization. Authors in [7] formulate optimal placing chained VNFs in an operator's network with multiple sites. They describe the mapping as a Mixed Integer Quadratically Constrained Program (MIQCP) to place and chain the network functions while considering the limited network resources and the requirements of the functions. In [26], authors study the energy-aware service function placement problem for SFC in data centers and formulate it as a binary integer programming (BIP). In [27], the authors investigate VNF placement problem for the optimal SFC design across geographically distributed clouds. They consider inter-cloud latencies and VNF response times, and set up the problem as an Integer Linear Programming (ILP) optimization. The main limitation of ILP, BIP and MILP approaches is the exponentially growing complexity with problem size. This is exactly the scalability problem that we aim to avoid through a selection process that limits the set of retained candidates to host the requests in order to scale with size. Table 1 summarizes the most relevant work dealing with the VNF placement and chaining problem. This table compares algorithms in terms of complexity, strengths and weaknesses and lists algorithms that support VNF sharing and use batch techniques to improve performance and resource utilization.

In summary, most of the existing works in the literature proposes heuristic algorithms ([13], [10], [11], [12], [2], [14]) to deal with large infrastructure instances and problem sizes. They however do not solve optimally the VNF placement and chaining problem. Other works focus on optimal solutions and use MILP [3], [12],[22], [23], [24], [2], [8] [25], MIQCP [7] or BIP [26] algorithms to solve the problem. These algorithms are efficient in finding exact solutions for small problem sizes and are not suitable for data centers with thousands of

physical machines.

Our proposal is based on an ILP model with a selection process that reduces the number of candidates in order to scale much better with increasing problem size. In addition to that, our proposed algorithms consider sharing of VNF resources between different VNF-FGs/chains and consider handling the requests in batch mode to enhance performance. Our model is also sufficiently generic to support the optimization of different criteria. Referring to Table 1, and especially the VNF sharing column, the MCTS algorithm supports VNF sharing to improve resource usage, reduce provider cost and increase revenue as well as energy efficiency objective. Consequently, we compare our proposed algorithm with MCTS. MCTS has already been shown to perform among the best proposed algorithms to address the placement and chaining of VNFs and this makes in our view the selected comparison legitimate and meaningful. We present next our modeling framework followed by the performance of our proposed solutions.

## 3. Problem formulation and mathematical model

This section presents the modeling framework for the VNF placement and chaining problem and the mathematical model and criteria used for deriving the ILP based algorithms.

### *3.1. Network Model*

The substrate or physical network, called by the NFV-I by ETSI [28], is modeled as an undirected weighted graph, denoted by $G = (V(G), E(G))$ where $E(G)$ is the set of its physical links and $V(G)$ is the set of the physical nodes.

Each physical node, $w \in V(G)$, is characterized by its i) available resource denoted by $C(w)$ (representing the node remaining capacity in compute, memory or storage resources), and ii) type $T(w)$: switch, server or Physical Network Function (PNF) [29], where PNFs are the traditional physical middleboxes offering network functions. A PNF is a dedicated hardware that implements a network function.

Without loss of generality we consider primarily computing resources for $C(w)$. Adding resource types (e.g., memory, storage) is straightforward in our proposed mathematical model. A physical link $e \in E(G)$ is characterized by its available bandwidth $C(e)$.

The VNF-FG graphs requested by the clients are represented mathematically by a graph $D = (V(D), E(D))$ where $V(D)$ is the set of VNF-FG virtual nodes and $E(D)$ is the set of virtual links. Each virtual node, $v \in V(D)$, is characterized by its i) required processing capacity $C(v)$ and ii) its type $T(v)$: switch (i.e., ingress or egress) or VNF. Each virtual link $d \in E(D)$ is described by its required bandwidth $C(d)$. Note that we can also consider instead the end to end latency if this is the criterion to be taken into account. For this work, with no loss of generality, we consider the bandwidth as the client requirement. Our model is in fact generic and can be adjusted based on the client criteria and requirements. Figure 1 depicts an example of two VNF-FG requests (or two VNF chains in this case) comprising switches and VNFs to embed and deploy in the NFV-I. Typical examples of a VNF are: a firewall, a DPI, an SSL, a load balancer, a NAT, etc.

The NFV-I depicted in Figure 1 contains two PNFs, some interconnected servers and two switches (Switch1 and Switch2) serving the ingress and/or egress flows in the VNF-FG topologies. Each VNF requires a number of CPU units to operate properly and perform at some desired QoS.

In Figure 1, we assume for example that the firewall requires 15 CPU units, the NAT 10 CPU units and the DPI 8 CPU units for adequate VNF operation and performance.

Once any of these VNF instances are spawned, if shared, they can serve requests across multiple tenants up to their capacity limits. For a firewall instance, 15 CPU units are needed from the hosting physical node. Once deployed, the firewall instance, for example, can serve firewall function requests whose cumulative requirements do not exceed the 15 CPU units limit.

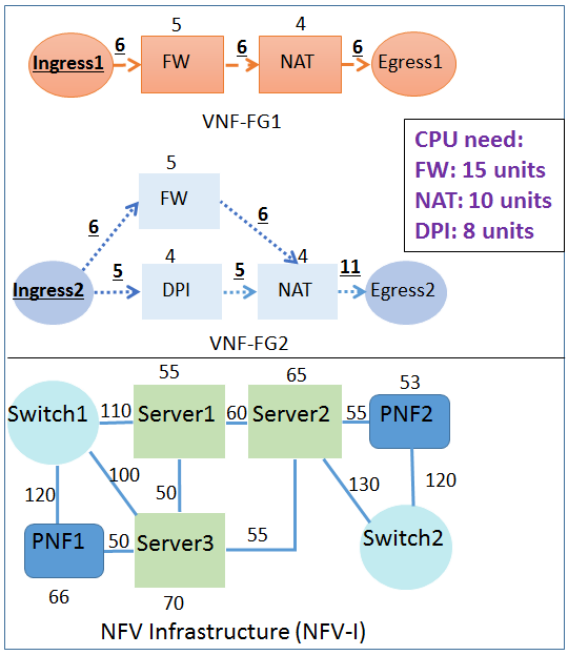


Figure 1: An example of SFC and NFV-I topologies

Table 1: Optimization algorithms

| Algorithm | Main idea | Complexity | Strength | Weakness | VNF sharing | Batch |
|---|---|---|---|---|---|---|
| MILP [3] | Mathematical model<br>One shot | High<br>(NP-Hard) | Optimal solution | Scalability | No | No |
| Greedy [15] | Iterative<br>Local search<br>Backtracking<br>Two Stages | $O(\|V(G)\|^5 \cdot log(\|V(G)\|))$ | Locally exhaustive<br>Fast in some cases | Poor scalability<br>Local optima | No | No |
| Multi-Stage [2] | Multi-stage and Viterbi algorithm | High (not evaluated) | Efficiency of Viterbi algorithm | Time to build multi-stage graph<br>Small scenarios | Yes | No |
| MCTS [4] | Projection to Decision Tree Search | $O(I \cdot \|Cand\| \cdot\|V(D)\| \cdot\|V(G)\|^2)$<br>I, $\|Cand\|$: parameters | Low complexity | Multiple settings to calibrate the effcient process | Yes | No |
| DP [16] | Dynamic programming<br>Decomposition into subproblems<br>One shot | $O(\|V(G)\|^3)$ | Reasonable complexity<br>Scalability could be managed | Memory hungry | No | No |
| Eigen [15] | Similarity or pattern match<br>Analytic Centrality<br>One shot | $O(\|V(G)\|^3)$ | Mesh graphs | Loosely connected graphs<br>Decomposition time | No | No |

### *3.2. Energy consumption Model*

The power consumption of a physical machine $w \in V(G)$ at time $t$ is estimated as in [30]:

$$P_t(w) = P^i(w) + (P^M(w) - P^i(w)) \cdot U_t(w) \quad (1)$$

where $P^i(w)$ is the idle state power consumption of machine $w$ and $P^M(w)$ is the maximum power consumption of a fully used physical machine. $U_t(w)$ (between 0 and 1) is the proportion of used CPU capacity.

The main objective is to minimize the total consumed power $P_t(G)$ (or equivalently to maximize the total saved power ${P_t}^S$):

$$P_t(G) = \sum_{w \in V(G)} P_t(w) \quad (2)$$

### *3.3. Embedding and Chaining problem*

To describe the addressed embedding and chaining optimization problem, we first summarize the related valid conditions and imposed constraints on nodes and links.

Each virtual node $v \in V(D)$ may be embedded in a substrate node, $w \in V(G)$, that has enough physical resources. Formally, $C(w) \geq C(v)$. Note that virtual switches (i.e., ingress or egress) are mapped only onto physical switches and also that a VNF may be embedded in a PNF or a physical server based on the ETSI recommendations [28].

We illustrate these facts in Figure 2 that depicts the embedding of a forwarding graph (VNF-FG1) highlighted by the dashed lines, starting from ingress $switch1$, crossing the firewall and the NAT, and ending at egress $switch2$. Since we are primarily aiming at a model that can optimize the embedding and chaining of services and forwarding functions taking into account both the interests of the tenants and providers, we seek a model (an ILP model) that can handle constrained VNFs (when they belong to the tenants) and VNFs that can be shared across tenants (when the VNFs belong to the providers and are not constrained by the tenants).

When the VNFs are made available by the providers, they can be time shared to avoid the deployment of additional VNFs in order to reduce cost of hosting and induced physical energy consumption.

Figure 3 illustrates this sharing by describing two

client requests served by the same provider NAT function. After the mapping of the first request (VNF-FG1), it is possible for the second request (VNF-FG2) to share the NAT, that acts finally as a common VNF for the two clients. Note that for this example, the residual CPU capabilities of Server3 remain unchanged after the mapping of VNF-FG2 despite the embedding of the NAT for the second request. This is achieved by sharing the NAT by the two requests VNF-FG1 and VNF-FG2 that enables providers to optimize the use of their physical resources and reduce their global power consumption by switching off the maximum number of physical machines (PNFs and servers).

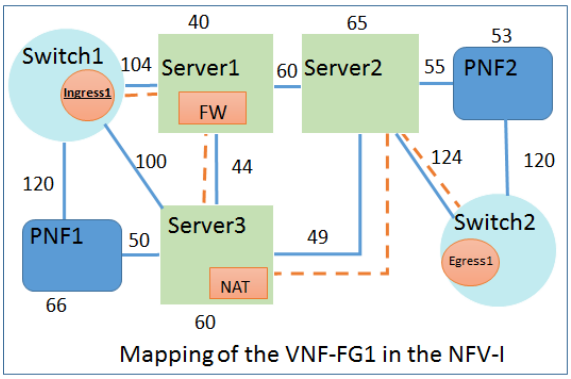


Figure 2: After the mapping of the VNF-FG1 in the NFV-I

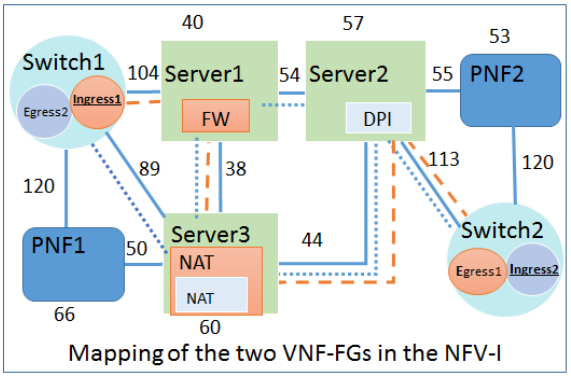


Figure 3: After the embedding of the two VNF-FGs

## 4. Online and batch algorithms

We propose a generic and comprehensive ILP to address the VNF-FG placement and chaining problem. The solution takes into account multiple criteria and whether VNFs are shared or not among several tenants. The model can be tuned to favor load balancing, consolidation, cost minimization, acceptance rates maximization or energy consumption minimization according to the goals and preferences of the stakeholders and their business objectives. The proposed ILP solution can operate online and respond to VNF-FG placement and chaining requests sequentially or in batch mode (with a batch of appropriate size) to treat a group of requests jointly. The goal of our investigation is to derive guidelines on adequate batch parameters to improve performance (compared to online and sequential operation) and to come closer to optimal solutions. To accomplish the placement and chaining of the VNFs as tenant requests arrive (or are treated in batch mode), both the requests themselves and the available resources (candidates for hosting) must be represented. The physical infrastructure $G$ has in practice a finite and limited amount of resources in nodes and links, and is consequently unable to host simultaneously a large number of VNF-FG requests. The objective for any optimization algorithm is typically to minimize the number of rejected requests while using the minimum amount of resources, respecting user requirements to ensure satisfaction and maximizing revenue for providers. The way requests are processed and handled by the optimization algorithm consequently matters and affects performance and overall optimization. Our objective is to explore via the proposed ILP based algorithms how such improvements can be achieved by online as well as batch mode placement and chaining. Both approaches will exhibit some strength provided they are judiciously configured and calibrated. Our goal is also to identify conditions more favorable to each approach.

### *4.1. ILP model for the online VNF-FG placement and chaining problem with VNFs sharing support*

As mentioned earlier, the NFV-I status needs to be updated to reflect the placement of previous VNF-FGs by removing already allocated physical resources. This includes the updating of remaining resources in physical hosts as well as shared (reused) VNFs that keep serving additional requests as long as they have remaining processing capabilities (e.g. room for time sharing of provider provisioned VNFs).

Figure 4 depicts this evolution as the placement of VNFs takes place as a graph extension after the mapping of a previous VNF-FG (VNF-FG1 for the example presented in Figures 1 and 2) and before handling the next VNF-FG request (VNF-FG2 of Figures 1 and 3). This graph extension, just after placement of VNF-FG1, is used for the placement of the next VNF-FG request (VNF-FG2), as illustrated in Figure 4. vspace-0.5cm The sharing of VNFs is taken into account by extending the infrastructure graph as depicted in Figure 4 where two fictitious nodes are added to the NFV-I following the placement of VNF-FG1. These fictitious nodes are added to the hosts, initially selected by the embedding

Table 2: Main notations

| Notation | Description |
|---|---|
| $V(w)$ | Set of virtual nodes that can be embedded in the physical node $w$ |
| $W(v)$ | Set of potential physical nodes for hosting the virtual node $v$ |
| $F$ | Set of all the fictitious nodes |
| $C(f)$ | Remaining capacities to be shared and consumed by other VNFs |
| $F(w)$ | Set of fictitious nodes directly connected to the physical node $w$ |
| $V(f)$ | Set of virtual nodes that can be served by the fictitious node $f$ |
| $x_{vw}$ | A binary variable indicates whether the virtual node $v$ is embedded in the physical node $w$ |
| $y_{vf}$ | A binary variable indicates whether a fictitious node $f$ is embedded in the VNF $v$ |
| $\mathscr{P}$ | Set of all physical paths in graph $G$ |
| $u_{dp}$ | A binary variable equal to 1 if the virtual link $d$ is embedded on the physical path $p$ |
| $\delta_{ep}$ | A boolean parameter indicating if the physical link $e$ belongs to the path $p$ |

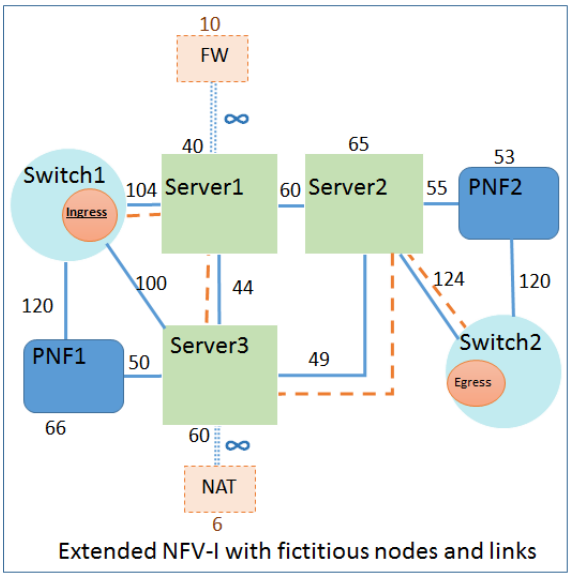


Figure 4: The extended NFV-I topology

algorithm of the VNF-FG1, as new neighbors. These nodes represent the remaining resources in the shared FW and NAT functions (original embedded in server 1 and server 3 during placement of VNF-FG1). Once the graph is extended, it reflects all remaining resources, and candidate nodes, available for potential hosting of new VNF-FG requests, VNF-FG2 in this example. The outcome of the ensuing placement is depicted in Figure 3 where the forwarding VNFs are placed in server 2 for the DPI and in server 3 for the NAT. The example depicts the NAT as a time shared VNF across tenants. Updating and extending the infrastructure graph (or NFV-I) allows us to take into account candidate physical nodes and links as well as candidate shared VNFs for hosting new placement and chaining requests. Our proposed ILP model for placing and chaining VNF-FG requests uses this graph extension as input in addition to the incoming VNF-FG requests. The notation adopted for the ILP is provided just before the ILP mathematical expressions (objective function, constraints and valid conditions) are presented:

- $W(v) \subseteq V(G)$ denotes the set of potential physical nodes (i.e., switches or servers and/or PNFs) for hosting the virtual node $v \in V(D)$ (i.e., ingress/egress or VNF) taking into account its requirements (i.e., processing power $C(v)$, type $T(v)$). Note that a virtual node, $v \in V(D)$, can be mapped to the substrate node, $w \in V(G)$, if: i) the available residual resources (i.e., $C(w)$) are at least equal to those required (i.e., $C(v)$), ii) node $v$ type is respected.
- $F$ is the set of all the fictitious nodes. Note that there are as many fictitious nodes per physical node as there are instances of shared VNFs in each hosting node.
- Each fictitious node $f \in F$ has a remaining capacities $C(f)$ that can be potentially shared and consumed by other VNFs.
- $F(w) \subseteq F$ is the set of fictitious nodes that are directly connected to the physical node $w$. In other words, $F(w)$ is related to VNFs that are already deployed in the physical node $w$ (see Figure 4 for more details). Each $f \in F(w)$ has a fictitious link connecting it to the $w$ with infinite capacity. In fact, the VNF is executed like an internal process into $w$.
- $F(v) \subseteq F$ is the set of fictitious candidate nodes that can serve/host the virtual node $v$. This is an analogous set to the physical candidates set $W(v)$.
- $V(w) \subseteq V(D)$ denotes the set of virtual nodes that can be embedded in the physical node $w$.

- By analogy, we define $V(f) \subset F$ that denotes the set of virtual nodes that can be served by the fictitious node $f$.
- $x_{vw}$ is a binary variable that describes whether the virtual node $v$ is embedded in the physical machine $w$ or not. Mathematically, this variable equals to 1 if the virtual node $v \in V(D)$ is assigned to the physical node $w \in W(v)$, 0 otherwise.
- $y_{vf}$ is a binary variable defined to enable the sharing of VNFs across (amongst) multiple tenants. As shown in Figure 4, we extend the initial physical topology by fictitious nodes that describe the remaining capacity of the already deployed VNFs to be used by other tenants. Beside the fictitious node, we also define the fictitious link that connects this new node to the physical node $w$ where the related VNF is running. Hence, $y_{vf}$ is describing whether the virtual node $v \in V(D)$ is assigned to the fictitious node $f \in F(v)$ or not. Variable $y_{vf}$ is defined for a fictitious node $f$ only if its corresponding VNF has the same type as $v$.
- $\mathscr{P}$ denotes the set of all physical paths in graph $G$. For each path $p \in \mathscr{P}$, $s(p)$ and $t(p)$ denote its end nodes. As we consider only elementary (loopless) paths, a path can be represented by the set of physical links (i.e., edges) it crosses.
- $u_{dp}$ is a binary variable equal to 1 if the virtual link $d \in E(D)$ is embedded on the physical path $p \in \mathscr{P}$.
- For each virtual link $d \in E(D)$, its end nodes are denoted by $a(d)$ and $b(d)$.
- $\delta_{ep}$ is a boolean parameter (not a variable) indicating if the physical link $e \in E(G)$ belongs to the path $p \in \mathscr{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in p$.

In addition to the model notation, the possibility of sharing VNFs across tenants and the constraints taken into account by the ILP model are now provided to extend the mathematical description of the placement and chaining problem. These descriptions are combined with the optimization objectives to complete the model. Each virtual node (VNF, ingress nodes, egress nodes) must be assigned to exactly one physical node (server, PNF, switch) or a fictitious node. Obviously, $F(w)$ is empty for the case of any physical switch $w$.

$$\sum_{w \in W(v)} \left( x_{vw} + \sum_{f \in F(w)} y_{vf} \right) = 1, \ \ \forall v \in V(D). \tag{3}$$

In this constraint, note that the fictitious nodes are considered as potential candidates just like physical nodes are. The capacity constraint for each physical node $w \in V(G)$ should be respected and not violated. In other terms, the total allocated virtual resources for a physical node should not exceed its remaining budget:

$$\sum_{v \in V(w)} x_{vw} \cdot C(v) \leq C(w), \ \ \forall w \in V(G). \tag{4}$$

By analogy, the remaining resources for each fictitious node $f \in F$ should not be violated while dealing with a new embedding. It is formally expressed as follow.

$$\sum_{v \in V(f)} y_{vf} \cdot C(v) \leq C(f), \ \ \forall f \in F. \tag{5}$$

Our ILP model supports optional co-localisation constraint when a provider imposes that two VNFs, $v_1, v_2 \in V(D)$ (i.e., issued from the same request), to be hosted in the same physical candidate for hardware, security or optimization reasons. The client (or tenant) is likely to express this kind of requirement to ensure a given quality of service or impose a geographic localization constraint. This constraint can be activated or disabled depending on stakeholders' requirements and interests. This constraint is formally expressed as:

$$x_{v_1 w} - x_{v_2 w} = 0, \ \ v_1, v_2 \in V(D), \forall w \in V(G). \tag{6}$$

By analogy, a provider or tenant separation constraint can be imposed and added to the ILP model so that two (or even more) VNFs, $v_1, v_2 \in V(D)$ (from the same request), are not embedded in the same physical node $w$. This constraint can be expressed using the product: $x_{v_1 w} \cdot x_{v_2 w} = 0$, that can be linearized by writing the constraint as an equivalent inequality:

$$x_{v_1 w} + x_{v_2 w} \leq 1, \ \ v_1, v_2 \in V(D), \forall w \in V(G). \tag{7}$$

The provider and/or the tenant will require whenever appropriate for them that any two virtual nodes belonging to the same request $D$ cannot be assigned to the same physical node. The constraint that a physical node $w$ may not host two different virtual nodes from the same request $D$ is expressed as:

$$\sum_{v \in V(w)} x_{vw} \leq 1, \ \ \forall w \in V(G). \tag{8}$$

Constraints 6, 7 and 8 are optional and may be activated or deactivated based on the provider to tenant agreements. The constraint of embedding each virtual link, $d \in E(D)$ in one single substrate path $p \in \mathscr{P}$, is expressed using the following equality:

$$\sum_{p \in \mathscr{P}} u_{dp} = 1, \ \ \forall d \in E(D). \tag{9}$$

If $p \in \mathscr{P}$ is the path selected to host the virtual link $d \in E(D)$ (i.e., $u_{dp} = 1$), then the path end nodes, $s(p)$ and $t(p)$, must be the substrate nodes where the two virtual extremities (i.e., virtual nodes) $a\,(d)$ and $b\,(d)$ of the virtual link $d$ are assigned to. Using the introduced binary variables $x$, $y$ and $u$ this constraint is:

$$\begin{aligned} & u_{dp} \leq \left(x_{a(d)s(p)} + \textstyle\sum_{f \in F(s(p))} y_{a(d)f}\right) \\ & + \left(x_{a(d)t(p)} + \textstyle\sum_{f \in F(t(p))} y_{a(d)f}\right), \\ & \forall d \in E(D),\ \forall p \in \mathscr{P} \\ & u_{dp} \leq \left(x_{b(d)s(p)} + \textstyle\sum_{f \in F(s(p))} y_{b(d)f}\right) \\ & + \left(x_{b(d)t(p)} + \textstyle\sum_{f \in F(t(p))} y_{b(d)f}\right), \\ & \forall d \in E(D),\ \forall p \in \mathscr{P}. \end{aligned} \tag{10}$$

The first inequality implies that if $u_{dp}$ is equal to one (i.e., the virtual link $d$ is embedded in the physical path $p$) $a(d)$ has to be embedded in one of the two extremities of $p$ (i.e., $s(p)$ or $t(p)$) or in one of their attached fictitious nodes. The second inequality indicates that if the variable $u_{dp}$ is equal to one, $b(d)$ should be mapped in either $s(p)$ or $t(p)$ or in one of the fictitious nodes connecting to them. If the right side of the inequality is equal to 0, the $u_{dp}$ variable must have a value of 0. In summary, if the two extremities of the virtual link $d$ are not embedded within the endpoints of the physical path $p$, it is impossible that $d$ is mapped in $p$.

The constraint 10 should be enough for the integrity of our proposed ILP model if the constraint 8 is activated (i.e., two virtual nodes belonging to the same request $D$ can not be embedded in the same physical node). However if the constraint 8 is deactivated, our proposed ILP needs an additional sanity constraint to maintain integrity. The issue generated by the deactivation of the constraint 8 can be described by considering two virtual nodes $v_1$ and $v_2$ connected by the virtual link $d$, a given physical path $p$ and assuming that the two virtual nodes $v_1$ and $v_2$ were colocated/embedded in $s(p)$ for instance. So the two right sides in the constraint 10 are equal to 1. Hence, for this case the $u_{dp}$ variable may have the value of 1, which means that the two extremities (i.e., $v_1$ and $v_2$) of the virtual link $d$ are in the two extremities of the physical path $p$. But, we already assumed that the two virtual nodes $v_1$ and $v_2$ are colocated in $s(p)$. We consequently need another sanity constraint to avoid this problem. This constraint should prevent that if $v_1$ and $v_2$ (i.e., which are connected by the virtual link $d$) are mapped in the same extremity of a given path $p$, the $u_{dp}$ variable is set to 1. Consequently, if $x_{a(d)s(p)}$ and $x_{b(d)s(p)}$ are equal to 1, $u_{dp}$ should be set to 0 and in addition if $x_{a(d)t(p)}$ and $x_{b(d)t(p)}$ are set to 1, $u_{dp}$ can not have 1 as a value. We can ensure this sanity condition by making sure the sum of these three variables does not exceed the value of 2. This constraint may be expressed after taking into consideration the fictitious nodes:

$$\begin{aligned} & u_{dp} + \left(x_{a(d)s(p)} + \textstyle\sum_{f \in F(s(p))} y_{a(d)f}\right) \\ & + \left(x_{b(d)s(p)} + \textstyle\sum_{f \in F(s(p))} y_{b(d)f}\right) \leq 2, \\ & \forall d \in E(D),\ \forall p \in \mathscr{P} \\ & u_{dp} + \left(x_{a(d)t(p)} + \textstyle\sum_{f \in F(t(p))} y_{a(d)f}\right) \\ & + \left(x_{b(d)t(p)} + \textstyle\sum_{f \in F(t(p))} y_{b(d)f}\right) \leq 2, \\ & \forall d \in E(D),\ \forall p \in \mathscr{P}. \end{aligned} \tag{11}$$

Equation 11 has to be considered only if the equation 8 is deactivated. The total consumed bandwidth (i.e., traffic load on the network) in each physical link $e \in E\,(G)$ should not exceed the physical link capability. This constraint is expressed as follows:

$$\begin{aligned} & \textstyle\sum_{d \in E(D)} C\,(d) \cdot \left(\sum_{p \in \mathscr{P}} \delta_{ep} \cdot u_{dp}\right) \leq C\,(e)\,, \\ & \forall e \in E\,(G) \end{aligned} \tag{12}$$

For a given VNF-FG request, our objective is to find an optimized mapping that minimizes, for instance, the total NFV-I power consumption. If the central criterion is energy saving, we hence minimize the cost of the mapping request based on the power consumption. Note that the value $P_t(w)$ is calculated regardless the current request in order to avoid any non-linearity problem.

$$\begin{aligned} \text{minimize } & \textstyle\sum_{v \in V(D)} \sum_{w \in W(v)} \\ & \left(x_{vw} + \textstyle\sum_{f \in F(w)} y_{vf}\right) \cdot P_t(w) \end{aligned} \tag{13}$$

We should highlight that our ILP model works fine with any objective function required by the provider whether it is related to minimizing the total consumed bandwidth or load balancing the datacenter or consolidating the machines or any other rational objective. Not forgetting that the optional constraints 6, 7 and 8 can be associated with the main ILP objective function on a need basis, we can summarize the ILP central objective, the constraints and obligations in the following set of equations:

$$\text{minimize} \sum_{v \in V(D)} \sum_{w \in W(v)} \left( x_{vw} + \sum_{f \in F(w)} y_{vf} \right) \cdot P_t(w)$$

subject to:

$$\sum_{v \in V(w)} x_{vw} \cdot C(v) \leq C(w), \forall w \in V(G)$$

$$\sum_{v \in V(f)} y_{vf} \cdot C(v) \leq C(f), \quad \forall f \in F$$

$$\sum_{w \in W(v)} \left( x_{vw} + \sum_{f \in F(w)} y_{vf} \right) = 1, \quad \forall v \in V(D)$$

$$\sum_{p \in \mathscr{P}} u_{dp} = 1, \quad \forall d \in E(D)$$

$$u_{dp} \leq \left( x_{a(d)s(p)} + \sum_{f \in F(s(p))} y_{a(d)f} \right) + \left( x_{a(d)t(p)} + \sum_{f \in F(t(p))} y_{a(d)f} \right), \quad \forall d \in E(D), \forall p \in \mathscr{P}$$

$$u_{dp} \leq \left( x_{b(d)s(p)} + \sum_{f \in F(s(p))} y_{b(d)f} \right) + \left( x_{b(d)t(p)} + \sum_{f \in F(t(p))} y_{b(d)f} \right), \quad \forall d \in E(D), \forall p \in \mathscr{P}$$

$$u_{dp} + \left( x_{a(d)s(p)} + \sum_{f \in F(s(p))} y_{a(d)f} \right) + \left( x_{b(d)s(p)} + \sum_{f \in F(s(p))} y_{b(d)f} \right) \leq 2, \quad \forall d \in E(D), \forall p \in \mathscr{P}$$

$$u_{dp} + \left( x_{a(d)t(p)} + \sum_{f \in F(t(p))} y_{a(d)f} \right) + \left( x_{b(d)t(p)} + \sum_{f \in F(t(p))} y_{b(d)f} \right) \leq 2, \quad \forall d \in E(D), \forall p \in \mathscr{P}$$

$$\sum_{d \in E(D)} C(d) \cdot \left( \sum_{p \in \mathscr{P}} \delta_{ep} \cdot u_{dp} \right) \leq C(e), \forall e \in E(G)$$

$$x_{vw}, y_{vf}, u_{dp}: \quad \text{binary variables}$$

### *4.2. ILP complexity and candidates selection*

The placement and chaining solution can be found using an ILP solver (e.g., Cplex [31]) for small NFV-I instances. Since the problem is NP-hard and does not scale with problem size, we resort to heuristic based solutions or modified ILP algorithms to find solutions faster. If the size if NFV-I is large, a judicious selection of hosting candidates can be used to reduce drastically computational complexity and resolution time. Reference [32] has shown that complexity of this kind of problem is $O(n^{3.5})$ if the variables $n$ are **continuous**. Where $n = O(|E(G)| \cdot |V(G)|)$ for the addressed placement and chaining problem. The case of continuous variables corresponds to a non feasible solution where one VNF can be split and mapped onto many candidates. This can be considered as **a lower bound** for the complexity of the NP-Hard VNFs placement and chaining problem. For worst cases of fully connected networks, we have: $|E(G)| = |V(G)| \cdot (|V(G)| - 1)/2$ and $n = O(|V(G)|^3)$ and accordingly face a complexity of $O(|V(G)|^{10.5})$ for the linear program, with continuous variables, that is governed by the NFV-I size (i.e., the number of physical nodes).

**Algorithm 1:** R-ILP algorithm

```
1 Inputs: G (NFV-I), D (VNF-FG)
2 Output: Mapping of the incoming request
3 Cands ← CandidatesSelection(G, D, k1)
4 for (d ∈ E(D)) do
5     for c1 ∈ W(a(d)) do
6         for c2 ∈ W(b(d)) do
7             p ← computePaths(c1, c2, k2)
8             P ← P ∪ p
9 CplexSolver(Cands, P)
```

The first step to reduce problem complexity is to select only feasible candidate physical nodes from the physical infrastructure based on their characteristics, available resources and provided hosting services. Prior to any optimization, we apply a deeper selection strategy that chooses a reduced number of prominent candidates, those that meet the virtual node $v$ computing and networking requirements. This set of candidates is denoted by $Cand(v) \subset W(v) \cup F$. We define also $Cands$ as the union of all $Cand(v)$ for each $v \in V(D)$. Formally, we have, $Cands \leftarrow \bigcup_{v \in V(D)} Cand(v)$. This additional pruning step discards all physical nodes that do not meet these requirements. We also define a maximum number, $k_1 = |Cand(v)| \; \forall v \in V(D)$, of eligible candidates to retain so as to reduce further the size of the set of the most prominent candidates. Tuning $k_1$ requires exploration and analysis of the impact of i) NFV-I size, ii) efficiency of the machine that runs the ILP solver and iii) the acceptable provider delay before providing a solution to the tenants' requests. This analysis is presented in Section 5.

Our R-ILP strategy is summarized in Algorithm 1 and is initialized in Step 3. Steps 4 to 6 search for the $k_1$ most prominent candidates among the physical and/or fictitious nodes represented in the model. For each virtual link $d$, we compute $k_2$ shortest paths, between the link extremities (i.e., $a(d)$ and $b(d)$), for each couple of physical and/or fictitious candidates. We make use of the Dijkstra algorithm with a new metric that promotes the paths with maximum remaining bandwidth. Steps 7 to 11 are dedicated to these paths computations. In step 12, we define the decision variables (i.e., $\{x_{vw}\}$, $\{y_{vf}\}$ and $\{u_{dp}\}$) based on the computed candidates (i.e., nodes and links) and solve the linear program using Cplex solver.

Algorithm 2 describes this candidates selection procedure. In step 6, all the potential candidates having sufficient computing and networking resources to host the VNF $v \in V(D)$ are extracted from the set of all physical nodes $V(G)$ and the set of fictitious nodes $F$. These

potential candidates are saved in a temporary set $P(v)$. During step 7, we make use bipartite graph matching [33]. In the terminology of bipartite graphs, a matching is a set of pairwise disjoint edges that link 2 sets of nodes. The advantage of using this method is that it finds the best candidate to host a given VNF based on our customized energy consumption metric. This bipartite graph matching to perform our selection provides us with the best matching between each VNF, egress and ingress. We use the bipartite graph matching algorithm on the two sets of nodes depicted in Figure 5. The first set contains all VNFs, the ingress and the egress nodes ($V(D)$). The second set includes all physical and fictitious nodes ($V(G) \cup F$). We weighted the links between these two sets by a cost function that computes the consumed energy if a given virtual a node $v$ is embedded within a physical node $w$. To define this cost function (i.e. $cost_e$ in algorithm 3), we make use of the formula $P_t(w)$ taking into account the required resources $C(v)$ of the virtual node $v$. The weight of a link connecting a fictitious node to a virtual node is set to 0, because there is no any energy to be consumed when a fictitious node is serving a VNF. Once these two sets are created and the links are weighted based on our cost function, we run the algorithm 3 that computes the best matching between the two sets. It actually finds the suitable and best candidate that may host each one of the virtual nodes based on the links weights. The bipartite graph matching algorithm ends by setting only the best candidate and not a subset of candidates as performed our algorithm R-ILP. Accordingly, we re-run the same bipartite graph matching algorithm after setting to infinity as the new weight value on each link selected in the previous iteration(s)/loop(s) in order to avoid reselecting the same candidate (step 9 of algorithm 2). This process is repeated $k1$ times to get at maximum $k1$ candidates for each virtual node. In terms of complexity, the maximum bipartite matching algorithm is achievable in $O(|V(G)| \cdot |V(G)|)$. We repeat these steps for a number of candidates k1. This leads to a complexity that will not exceed $O(|V(G)| \cdot |V(G)| \cdot k1)$ time complexity.

During step 7, we select from the set $P(v)$ the candidates based on two criteria: i) the amount of residual computing resources in the nodes and ii) a pattern matching criterion. Since we aim at reducing energy consumption, we favor and push for physical nodes with minimum residual resources to achieve consolidation, avoid soliciting more servers and hence power off or put in idle mode more servers. Accordingly, we select a first subset from $P(v)$ to reduce overall power consumption. Then, we proceed to a "network topology oriented" pattern matching check between the virtual node $v$ and the physical candidates. We use the concept of node topology that describes the node with its links to neighboring

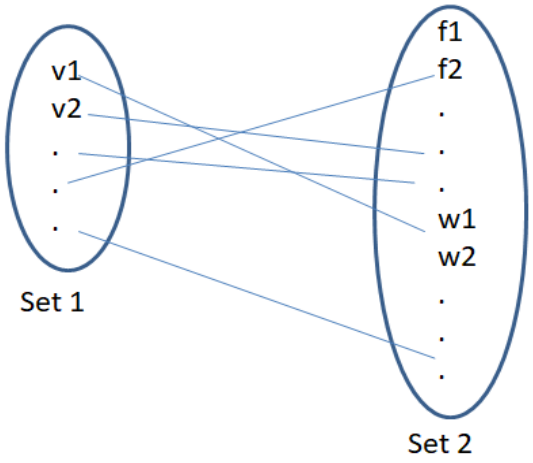


Figure 5: Bipartite graph

nodes (stronger notion than just node degree). We compare node $v$ topology to the topology of each selected candidate. The pattern matching between two topologies is based on minimum difference in number of connected links to a given node. There is full matching if this difference is 0. The bigger this difference the worse the pattern matching. In summary, the candidates with the best matches with $v$ (or with highest similarity or smallest difference in topology) are included in set $Cand(v)$ with respect to $k_1$.

**Algorithm 2:** CandidatesSelection

```
Inputs: G, D, k1
Output: Cands
Cands ← ∅
F ← computeAllFictitiousNodes(D, V(G))
Set2 ← V(G) ∪ F
E_G ← buildBipartiteGraph(V(D), Set2)
for i ≤ k1 do
    Cands ← Cands ∪ Max-Matching-
      Bipartite-Graph(V(D) ∪ Set2, E_G)
    updateLinksCost(E_G)
return Cands
```

The retained candidates by selectCandidates function are the fictitious nodes if there are any. If $k_1$ threshold is not reached after setting the fictitious nodes, some of the physical nodes are added. We promote the selection of physical nodes having the minimum of remaining resources to avoid switching on new servers and/or switches in order to minimize the entire energy consumption, which is our objective in this work. By selecting a reduced set of feasible and prominent candidates, according to Algorithm 2, the complexity lower bound of the Linear Program (LP), with continuous variables, will be lowered to a complexity level in line with the

**Algorithm 3:** Max-Matching-Bipartite-Graph algorithm

1 Inputs: bipartite graph $G = (set1 \cup set2, E_G))$
2 Output: M' = Mapping of set1 on set2
3 $M = \{e\}$, such that $e \in E_G$ is the edge with the minimum weight. $e$ is directed from $set1$ to $set2$;
4 Redirect all edges $e \in M$ from $set2$ to $set1$ with a weight $-cost_e$;
5 Redirect all edges $e'$ not belonging to $M$, from $set1$; to $set2$, with a weight $+cost_e$;
6 Let $set1_M = \{u \in set1 \setminus M\}$; and $set2_M = \{u \in set2 \setminus M\}$;
7 Find an existing shortest path $P$ from $set1_M$ to $set2_M$;
8 Put $M' = M \Delta P$ ($M \Delta P = (M \cup P) \setminus (M \cap P)$), and repeat from step 4;
9 $M'$ is the optimal matching.

algorithm desired capabilities and scalability. In fact, the number of variables $n$ depends on the size of the $\{x_{vw}\}$ and $\{y_{dp}\}$ variables: $n = |\{x_{vw}\}| + |\{y_{dp}\}|$. With the reduction, the size of $x$ variables is $|\{x_{vw}\}| = k_1 \cdot |V(D)|$. Since we consider $k_1$ candidates for each of the two extremities of a virtual link $d$, we define $k_1^2 \cdot k_2$ variables (i.e., for each virtual link $d$), where $k_2$ is the number of path candidates between two physical nodes. Accordingly, $|\{y_{dp}\}| = k_1^2 \cdot k_2 \cdot |E(D)|$ and $n = k_1 \cdot |V(D)| + k_1^2 \cdot k_2 \cdot |E(D)|$. In the case of the smallest client request, $D$ is a chain so $|E(D)| = |V(D)| - 1$. Hence, we conclude that $n = O(k_1^2 \cdot k_2 \cdot |V(D)|)$ and the complexity of our LP is $O(k_1^7 \cdot k_2^{3.5} \cdot |V(D)|^{3.5})$. In the worst case of a fully meshed network $D$, $|E(D)| = |V(D)| \cdot (|V(D)| - 1)/2$ and thus $n = O(k_1^2 \cdot k_2 \cdot |V(D)|^2)$. Consequently, our LP complexity in the worst case (for a fully meshed request) is $O(k_1^7 \cdot k_2^{3.5} \cdot |V(D)|^7)$. The complexity of the LP is consequently independent of the size of the NFV-I and is only impacted by the candidates selection and paths computation steps. The size of the NFV-I will ultimately not entirely govern the LP resolution (or execution time).

Based on this candidates limitation, the Cplex execution time (i.e., linear program solving) is shown to be independent of the NFV-I size while the complexity of other steps depends directly on the number of nodes in the infrastructure (or NFV-I). Most of the R-ILP algorithm execution time penalty lies in the first stages of the overall process and not in the Cplex resolution phase of the algorithm. This analytical result is confirmed by the simulation and performance evaluation results reported in Section 5. From this proposed new ILP based online algorithm, called R-ILP (since it uses a reduced set of candidate hosting nodes), that leverages the VNFs sharing concept, we define a batch strategy that operates on and analyzes simultaneously a group of VNF-FG requests to optimize their placement and improve performance further compared with the online R-ILP algorithm that treats sequentially the requests by processing them on first come first serve basis.

### *4.3. Batch placement and chaining algorithm*

The current state of the art focuses mainly on online VNF-FG requests embedding. Our objective is also to explore batch embedding and propose seminal work and analysis for this alternative approach to the VNF-FG placement and chaining problem. With the batch mode more client requests are likely to be accepted and served compared to the online mode. The number of accesses to the NFV-I in the batch mode is smaller. Mapping a batch of $n$ requests requires only one access to the NFV-I while $n$ accesses will be needed for the online mode. Operating in batch mode can potentially provide benefits, by handling multiple requests over a viable time interval, in terms of quality of the optimization with a likelihood of coming closer to optimal solutions. When operating online, the solutions can be instantaneously optimal but are typically suboptimal over longer time durations since there is no look ahead nor combined look at past, present and future requests. The online solutions have the advantage of not incurring additional delays in providing embedding solutions on a sequential basis, processing one request at a time. Operating in batch mode enables, for instance, the processing of multiple requests as a group or an equivalent composite request. The batch mode will lead to solutions closer to optimal for the group of requests assuming the size of the batch and the interval over which the group is composed are adequately selected. Providers can reduce the cost of placing and chaining the multiple requests and hence improve their revenues as long as service level agreements with the tenants are met. The batch mode should lead to better utilization of the provider physical infrastructure and the provider provisioned VNFs (that they can even share across tenants). The providers would logically have more freedom to improve or increase their revenues and at the same time minimize rejection rate of tenant requests. This should be beneficial to all stakeholders but we need to verify the conditions for such improvement compared to the online mode. In addition, the improvement must be significant to justify the implementation of the batch mode. The goal of our analysis and study is to set the stage for deeper investigations into the batch mode and facilitate the development of formal performance assessment models in the future.

At this stage, we conduct an initial study based on the batch embedding mode described in Figure 6 where there is a time window, denoted by $W_b$, during which the incoming requests are stored in a dedicated list denoted by $A(W_b)$. Arrivals and departures occur during such windows as depicted in the second batch window

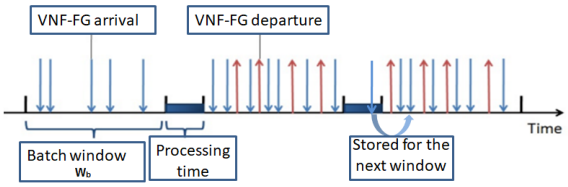


Figure 6: Processing of VNF-FGs in a batch mode

example in Figure 6. At the end of each batch window $W_b$ there is a required processing period during which the provider executes a management policy to decide for instance i) the embedding order of the already arrived VNF-FG requests (the stored requests in $A(W_b)$) and ii) the requests to be accepted in case there are insufficient physical resources to host all the requests in the batch. This should always be done to achieve highest provider revenue and maximizing the satisfaction of tenants by keeping rejection rates to a strict minimum, i.e. to the smallest possible value depending on available resources during the batch.

Our proposed batch oriented algorithm applies the R-ILP to the batch and takes into account the scalability of our online R-ILP algorithm by limiting the batch size (retaining only the best candidate hosts) to obtain good solutions in reasonable times. The algorithm handles the queued VNF-FG requests for batch processing in a heuristic way. Instead of trying to place optimally the entire batch $A(W_b)$ (i.e., the stored incoming VNF-FGs during the batch window $W_b$), and end up rejecting all queued VNF-FG requests when there are no optimal placement solutions for the entire batch, the algorithm processes the VNF-FG requests by analyzing the whole batch to determine the order in which the requests must be treated to meet a selected objective such as maximizing provider revenue, minimizing energy consumption, maximizing resource utilization, etc. In our performance assessment of the proposed batch based algorithm, we sort the VNF-FG requests in the batch queue $A(W_b)$ according to the revenue generated (foreseen) by their successful placement. The acceptance revenue $\mathbb{R}(D)$, of a given request $D$, is formally expressed as defined in [34][35]:

$$\mathbb{R}(D) = \sum_{v \in V(D)} \left(C\left(v\right) \cdot U_c\right) + \sum_{d \in E(D)} \left(C\left(d\right) \cdot U_b\right) \tag{14}$$

where $U_c$ is the realized gain from allocating one unit of CPU resources and $U_b$ is the earned revenue from the allocation of one bandwidth unit. Requests generating the largest revenues are placed (served) first in order to maximize overall achieved revenue and at the same time avoiding the rejection of all the requests at once when they are treated jointly in one shot. This placement strategy limits rejection rate while simultaneously aiming at maximizing the provider revenue. As long as the rejection rate is kept low, below $5\%$ for example, user satisfaction and quality of experience should remain acceptable.

**Algorithm 4:** Batch algorithm

```
Inputs: NFV-I, Batch window W_b
Output: Mapping of the incoming requests during W_b
A(W_b) ← ∅
while (W_b not expired) do
    if (arrival of new request D_i) then
        A(W_b) ← A(W_b) ∪ {D_i}
S(W_b) ← sort(A(W_b))
M ← ∅
while (S(W_b) not empty) do
    D ← extractTop(S(W_b))
    m ← R-ILP(NFV-I, D)
    if (m ≠ null) then
        M ← M ∪ {m}
retrun M
```

The batch algorithm, Algorithm 4, operates as follows: upon arrival each incoming request $D_i$ is stored in the batch list $A(W_b)$. At the expiration of the batch window $W_b$, the queued VNF-FG requests are sorted based on their (expected) revenue and stored in a new list $S(W_b)$. The ordered VNF-FG requests in $S(W_b)$ are processed sequentially by the online R-ILP algorithm. Successfully placed VNF-FG requests are stored in a list $M$ along with their placement solution. Requests that can not be placed are rejected. The placement process continues until all requests have been processed irrespective of the placement outcome in order to place as many requests as possible and thus reduce rejection rate. This is the simplest batch mode algorithm, called Batch R-ILP (BR-ILP) in our case. A more elaborate approach, or extension of this simple mode, is to store client requests $D$ that could not be placed in a given batch window and resubmit them in the next batch window. The non satisfied request is re-considered only once before it is definitely rejected (using a P-persistent mode with P=1). This persistent version of the batch algorithm is called Resubmit BR-ILP (RBR-ILP). RBR-ILP aims at minimizing rejection of requests further by giving "not yet" served client requests a higher chance of being accepted in ensuing batches.

## 5. Performance Evaluation

The evaluation focuses on assessing the performance improvements achievable when sharing VNFs (provider provisioned VNFs) across multiple tenants. The evaluation also analyzes and explores the benefits that batch processing of VNF-FG requests can provide. The per-

formance of our proposed algorithms (online and batch heuristic which are publicly available [36]) are also compared with the MCTS state of the art algorithm reported in [4] that already highlights the relevance of sharing virtualized functions. The impact of the number of selected candidates for hosting the VNFs on the ILP algorithm execution time and successful placement is also addressed. The analysis is conducted to also provide insight on the most relevant candidate set sizes to keep execution time low while realizing most of the potential performance improvement. Finally, the algorithms are compared in terms of consumed energy and accepted requests to assess the energy consumed on average per hosted request while integrating the acceptance (or rejection) rate in the assessment.

### *5.1. Simulation Environment*

The performance evaluation is realized using a system simulation accepting both the NFV-I topologies (with their available resources for hosting) and the VNF-FG requests (with their resource requirements) as inputs to the optimization algorithms that search for a placement solution provided as an output of the java-based simulation. Both the NFV-I and VNF topologies (or graphs) are generated using the GT-ITM tool [37] with NFV-I topology sizes in the range $[100, 500]$ nodes. The connectivity between nodes is set to $0.3$. The initial physical capacities (i.e., CPU and bandwidth) are set to $150$ CPU units and $100$ bandwidth units respectively. For our batch algorithm (denoted "Batch"), the batch window $W_b$ size is $100$ time units and each resource request has an exponential lifetime with a mean of $500$ time units. The VNF-FG request size varies from $10$ to $50$ nodes depending on the scenario used for performance evaluation. In each simulation $1000$ requests are generated with Poisson arrivals for the requests with a mean arrival rate $\lambda$ of $5$ requests per $100$ time units. The arrival rate $\lambda$ varies for scenarios that assess performance for variable system load. The service times are exponentially distributed with a mean of $500$ time units. The required capacities (in CPU and bandwidth) are fixed to $10$ CPU units and $10$ bandwidth units respectively. The results obtained from multiple simulations are averaged and presented with $95\%$ confidence interval on the reported results. Results illustrating the behavior of the algorithms as a function of simulation time have no reported confidence interval as they correspond to only one run. The simulations were conducted using a standard PC with an *i-7* Intel Core i7 with 2.70 GHz, $8$ Cores and $16$ GBytes of RAM.

### *5.2. Performance metrics*

The performance metrics we consider to perform simulations and to compare fairly our proposal with the other algorithms and assess performance are:

**1) Rejection percentage:** is the percentage of VNF-FG requests that have not been accepted due to unavailability of physical resources.

**2) Acceptance revenue:** is the service provider realized (generated) revenue (benefits) at time $t$ when accepting client requests (in our case successful placement of VNF-FG requests). The acceptance revenue is formally expressed as:

$$\mathbb{R}(t) = \sum_{D \in \mathbb{AR}_t} \mathbb{R}\left(D\right) \tag{15}$$

where $\mathbb{AR}_t$ is the set of the accepted requests and $\mathbb{R}(D)$ is the $D$ request gain as defined in equation 14. Note that $\mathbb{R}$ is the total revenue taking into consideration all the incoming requests.

**3) Execution time:** is the time needed to find an embedding solution for one VNF-FG request. This metric reflects the ability of the algorithms to scale with problem size and this is especially important for the ILP algorithm assessment.

**4) Resource usage:** This is the number of active servers (i.e., the number of nodes that are used to host the requests) divided by the total number of infrastructure nodes or physical servers. For machines that do not host any request, they consume only their idle power as expressed in equation 1 and this represents the minimum amount of local power consumed in each node. Saving energy would require shutting off machines to reduce consumption but the amount of energy needed to power back the machines needs to be taken into account for the evaluation to be complete. For the sake of simplicity and without any impact on the derived performance evaluation results, we assume that the machines enter only idle and power saving mode and also assume that the overall energy required in these states is far less than the required energy and power consumed by active nodes and that the consumed power is directly proportional to the compute and processing load on the servers.

**5) Power consumption:** is computed using equation 2. The global consumed power at time $t$ , $P_t(G)$, is the aggregation of the energy consumed by all the machines (i.e., servers, PNFs and switches) at this time $t$.

### *5.3. Comparison of our proposal to the exact resolution*

We evaluate the gap between the optimal solution provided by the unmodified and "Exact" solving our ILP model by exploring the entire search space (i.e., all the potential candidates) with i) our R-ILP algorithm operating on a selected and reduced candidates subset and ii) a competitor algorithm from the related work [3] denoted Competitor. To do so, we consider the **quality of the mapping** of each request as the evaluation and comparison criterion. Since we aim in this work at minimizing energy consumption, we define the quality of

the mapping based on the value of the total consumed energy $P_t(G)$ (c.f., equation 2) that describes the algorithm ability to reduce the overall energy consumption. Since the exact algorithm and the related work [3] (i.e., Competitor), do not scale with size, we can only perform the comparison for small NFV-I size. We report the results based on the **realistic** GEANT topology from SNDlib [38] including of 22 nodes and using 1000 VNF-FG requests composed of 5 VNFs. We also set the number of retained, selected, candidates to 5 to reveal the capabilities, possible strength, of the proposed algorithm that applies the original ILP objective function on this rather very small reduced set. The Exact (optimal) strategy considers all the potential candidates for each VNF. The gap, measured in percentage of different solutions compared with the Exact algorithm, is formally expressed as follow:

$$Gap(\%) = \frac{P_t(G)^{algo} - P_t(G)^{Exact}}{P_t(G)^{Exact}} \times 100 \quad (16)$$

$algo$ refers to one of the considered algorithms: i) R-ILP or ii) Competitor or iii) MCTS. In Figure 7, the gap does not exceed $10\%$ for our proposed R-ILP (operating on a selected reduced set of candidates, 5 in this evaluation) and $12\%$ for the Competitor strategy, respectively, compared to the Exact algorithm.

Note that R-ILP achieves the same performance as the exact algorithm in the time interval $[0, 220$ time units$]$ , the gap is equal to $1\%$, this section of the results is not reported on Figure 7 (black curve) whose ordinate is in logarithmic scale (logscale). MCTS is usually outperformed by the other strategies (the ILP based algorithms) with nevertheless a reasonable gap with respect to the exact algorithm. The maximum gap of $18\%$ between MCTS and the Exact algorithm occurs only for very few cases. R-ILP typically outperforms the Competitor algorithm with R-ILP curve (black) most frequently below the Competitor gap curve (orange). The R-ILP achieves this good performance (with very small gaps, sometimes even no gap, with the exact ILP) since VNFs are shared and a judicious selection of candidate hosts is operated prior to applying the ILP. This result confirms that using a judicious candidates selection process, the optimal solution can be found sufficiently often and the percentage of time R-ILP results are suboptimal kept below a certain percentage (below $10\%$). These encouraging results call for a deeper investigation on the efficiency of our proposed algorithm that applies the original ILP on only a reduced set of candidates.

### *5.4. Impact of the number of candidates*

To avoid the well known scalability problems of integer linear programs because of state explosion with increasing problem size (number of requests, sizes of requests and infrastructure sizes), our strategy consists in limiting the number of candidates for hosting VNFs to the smallest possible sizes, those that can find placement solutions in acceptable and practical execution times for the stakeholders. The selected number of candidates should neither be too small nor too high to ensure good quality of the solutions while keeping execution times low. The evaluation aims at finding the appropriate number of candidates to select and use for the VNF-FG placement and chaining, especially those that can provide good hosting solutions in reasonable execution times.

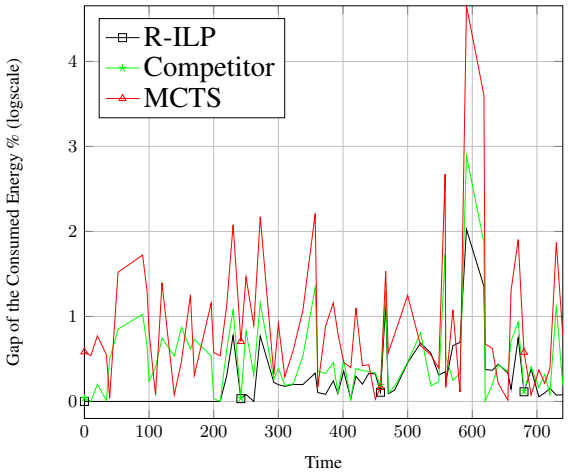


Figure 7: The gap between Exact and other strategies in percentage

To find these most adequate candidate set sizes, we vary the number of selected candidates among all feasible physical nodes for hosting the VNFs and measure the revenue achieved by the provider, the rejection rate of the VNF-FG requests and the execution time needed to find the placement solutions that affects directly quality of service and experience.

The results which are reported in Table 3 reveal the tradeoff between the size of the candidates subset and the quality of the mapping solution. The number of candidate paths (i.e., related to variable $u_{dp}$ in our ILP model) is fixed to 3 for these reported results that correspond to a high load scenario with $\lambda$ (i.e., mean arrival rate of requests) set to 25 per 100 time units. If the number of selected and used candidate is too low, e.g. for 5 candidates in Table 3, the percentage of requests that are rejected is high $(43.2\%)$ and unacceptable. Increasing the number of candidates will of course improve this metric and the quality of the solutions at the expense of execution times. The results of Table 3 confirm that there is a tradeoff between limiting the number of candidates and the execution times and this occurs when no additional benefits can be achieved by increasing the number of candidates. Beyond 10 candidates the performance improvements in rejection rate become

marginal. The simulation results show a rejection rate of 27.9% for 10 candidates that improves slightly to 26.6% for 20 candidates followed by a negligible improvement from 20 to 30 candidates (with almost the same rejection rate 26.6% versus 26.4%). The same can be observed for the provider revenue that improves marginally beyond a certain size of the candidate set. The execution time results confirm this fact more forcefully since the execution times for 20 and 30 candidates are considerably higher than the execution times for 10 candidates (2316.97 ms for 20 and 4154.20 for 30 candidates versus only 814.79 ms for 10 candidates). These results confirm that limiting the number of candidates for hosting does not degrade quality of the solutions, increases marginally rejection rate but can considerably reduce execution times and hence address the scalability problem of exact or integer linear programming solutions. This also indicates that it is not always justified to resort systematically to non ILP based heuristics to address VNF-FG placement and chaining.

### 5.5. Execution time and revenue evaluation

Figure 8 depicts the rejection ratio for a low load scenario when $\lambda = 5$. We notice that the R-ILP based algorithms (online and batch) outperform the MCTS strategy by accepting more requests. The MCTS algorithm can not serve 19.3% of the incoming requests however the online R-ILP rejects only 3.4%. This performance highlights the efficiency of our R-ILP algorithm. The rejection rate is reported as a function of simulation time with the curves highlighting a transient state before reaching a steady state condition for all the algorithms corresponding to the nominal load where and when the results are actually collected to produce all other performance assessments.

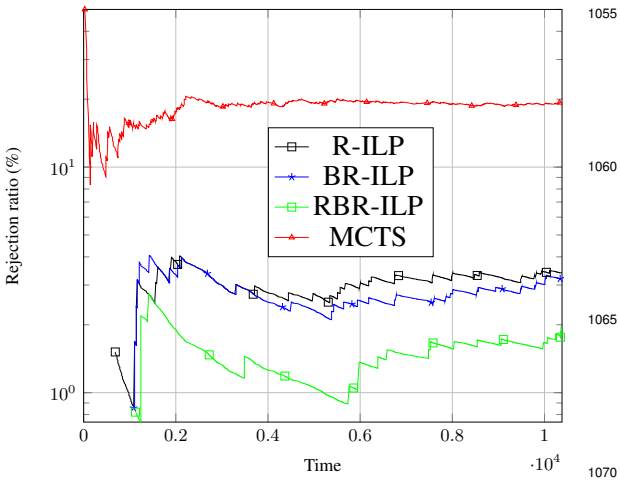


Figure 8: Rejection percentage for $\lambda = 5$

Figure 9 reports the rejection rate performance for increasing system load for our algorithms and MCTS. We recall that each request has an exponential lifetime with a mean of 500 time units. At lower load, for arrival rate $\lambda = 5$, the online and batch algorithms reject around 3% out of the 1000 simulated placement requests while the MCTS algorithms rejects already 19.3%. For higher loads, especially for $\lambda = 10$, the situation worsens to 45.9% for MCTS and a bit less for our algorithms to 29.0% for batch and 36.4% for the online. The batch version rejects 7% less than the online one since the batch algorithms process a group of requests jointly, favor the placement of requests that generate the highest revenue in the batch and place as many requests as possible from each batch.

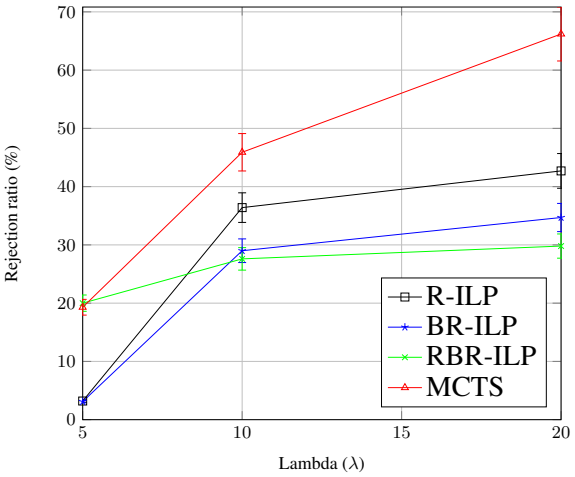


Figure 9: Rejection percentage w.r.t the mean of arrival rate $\lambda$

This superior performance of the batch algorithms is highlighted further by the results on the achieved revenue depicted in Figure 10. The MCTS is shown to provide much lower revenues while the online improves by a factor of 2.5 the revenue gains. The batch algorithms provide an additional gain of 10% for simple BR-ILP and 14% for RBR-ILP respectively beyond what the online can accomplish. The improvement factor compared to MCTS increases to 2.8.

The execution time of our R-ILP algorithm is assessed by varying the size of the NFV-I while fixing the number of candidates to 10 (based on the results reported in Table 3) and the size of the VNF-FGs to 10 (VNF-FG sizes typical do not exceed 10 VNFs per graph). Figure 11 reports the overall needed execution time as a function of NFV-I size when including all the R-ILP strategy steps and Figure 12 that measures only the Cplex execution time part of this overall time. This allows us to reveal that the execution of the R-ILP algorithm without all the other steps (generation of requests, preparation of the VNF-FG requests acting as

Table 3: Impact of the candidates number

| | 5 candidates | 10 candidates | 20 candidates | 30 candidates |
|---|---|---|---|---|
| Percentage of rejection | 43.2% | 27.9% | 26.6% | 26.4% |
| Revenue | 329508 | 437768 | 444724 | 448688 |
| Execution time (ms) | 172.29 | 814.79 | 2316.97 | 4154.20 |

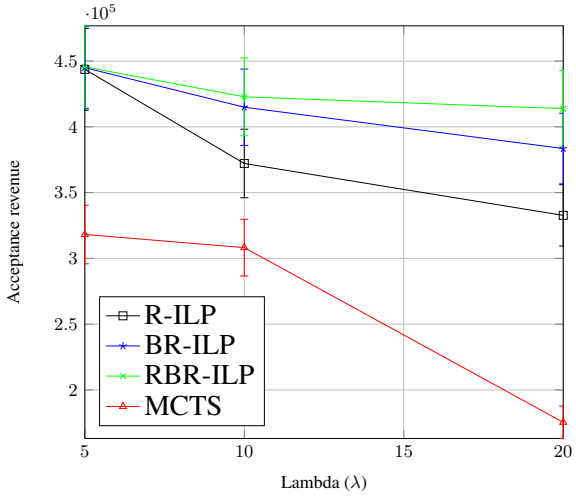


Figure 10: Acceptance revenue w.r.t mean of arrival rate $\lambda$

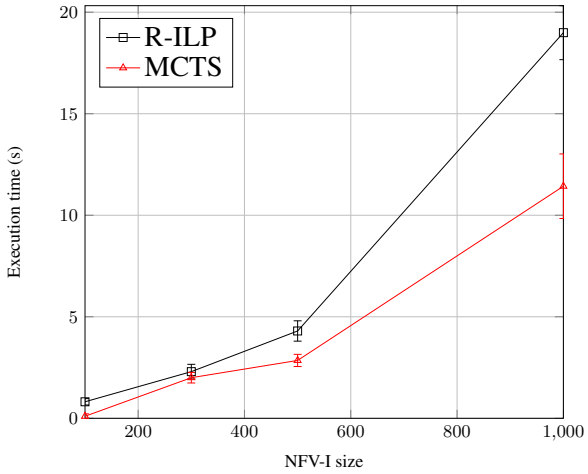


Figure 11: Total execution time w.r.t NFV-I size

inputs to the algorithm and updating of the NFV-I resources availability) is very low and is a marginal fraction of the overall execution time. Figure 11 reports a very low overall algorithm execution time for a NFV-I with 100 nodes that increases to 18.99 seconds for 1000 NFV-I nodes. These higher values remain nevertheless acceptable for an ILP program. Figure 11 also highlights the gap in execution time between our R-ILP algorithm (with a candidates size fixed to 10) and the MCTS heuristic that outperforms slightly the R-ILP. This gap remains low (about 7 seconds for 1000 NFV-I nodes) when analyzed jointly with the other performance metrics such as the achieved revenue by both algorithms (cf. Figure 10, where the R-ILP outperforms MCTS).The R-ILP execution time for 1000 NFV-I nodes of 18.99 seconds remains however competitive and a reasonable price to pay with the superior performance in all other metrics and in proximity to the ILP (see Fig. 7 ). The R-ILP outperfoms the MCTS heuristic also in rejection rate as depicted already in Figures 8 and 9 and in comsumed energy as reported later on in Fig. 18. Consequently, the R-ILP achieves and ensures the best tradeoff between earned revenue and execution time. Figure 13 depicts the overall and Cplex execution times in logarithmic scale against the NFV-I sizes to emphasize the previous findings. Namely to confirm more clearly that the Cplex execution time for the on line ILP and batch modes increases marginally with NFV-I sizes since it is dominated by the small (controlled) number of selected candidate hosts in the optimization. The overall execution time on the contrary has exponential increase because of the exponentially growing complexity of most of the steps in Algorithm 1. The gap between these times is at least one order of magnitude for small NFV-I sizes ($10^2$ versus $10^3$) for NFV-I with 100 nodes. The gap grows to two orders of magnitude for 1000 nodes ($10^2$ versus $10^4$). These simulation results confirm the complexity analysis reported in Section 4. Figure 14 evaluates the performance of

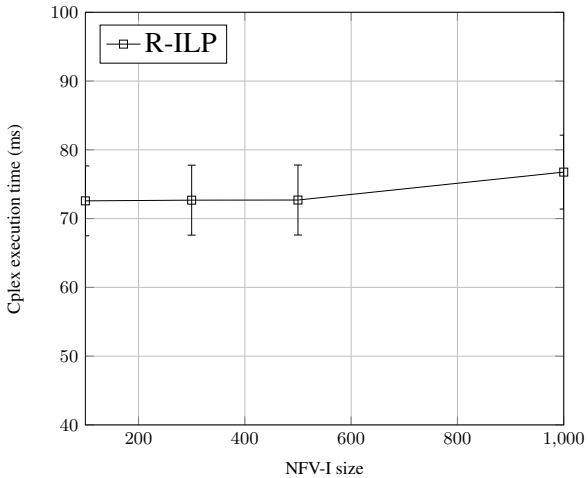


Figure 12: Cplex execution time w.r.t NFV-I size

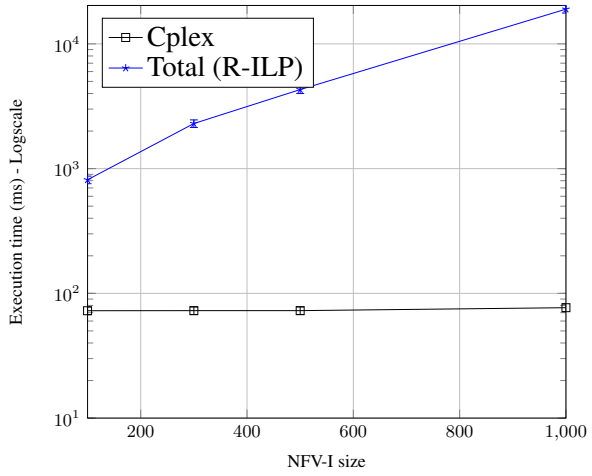


Figure 13: Cplex Vs total execution time w.r.t NFV-I size

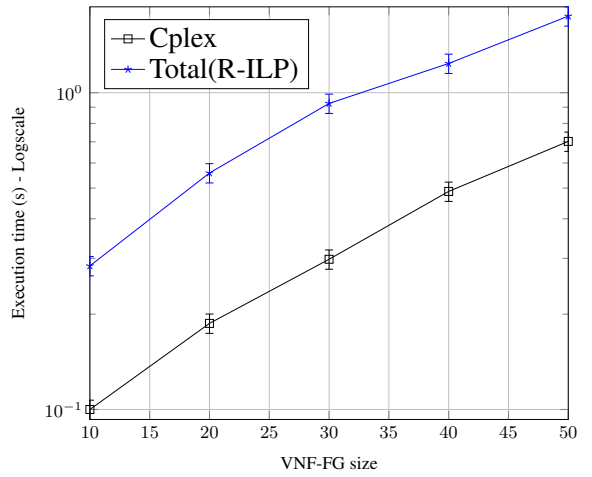


Figure 14: Total Vs Cplex execution time w.r.t VNF-FGs size

the R-ILP as a function of increasing VNF-FG request sizes (in a logscale way) for the overall R-ILP execution time and the execution time of Cplex step only. We fix the NFV-I size to 100 nodes to calibrate the evaluation in comparison to the sizes of the VNF-FG requests that span sizes from 10 to 50 nodes so the assessment can reveal the effect of VNF-FG request sizes on the R-ILP speed. Figure 14 shows the evolution of the overall R-ILP algorithm execution time for increasing VNF-FG sizes and reports an overall execution time that remains very low (below 1.8 seconds) even for VNF-FG sizes of 50 nodes. The slope of the evolution curve is not that steep (0.03) and the overall performance is acceptable for such requests sizes. Figure 14 focuses, also, on the required time to accomplish the Cplex processing step (step 12 of Algorithm 1). The progression of this execution time is also roughly linear with increasing VNF-FG size. We remark that the progression of the execution time is roughly linear (between 0.1 and 0.7 second) even if a bit steeper for larger VNF-FG sizes. This is expected since the Cplex solver is faced with a harder problem (including more variables) to solve for larger requests and a comparatively rather small NFV-I. By fixing the number of candidates the Cplex will be essentially faced with the same number of variables and the execution time will remain mostly constant as observed previously in Figure 12 with a variation in execution time not exceeding 4.08 ms for the evaluated range (from 72.67 ms to 76.75 ms for the 100 to 1000 range in NFV-I size). However, when the size of the VNF-FGs increases, the Cplex step is faced with an increasing number of $x_{vw}$ ($y_{vf}$ also) variables and consequently increasing $u_{dp}$ variables and constraints. This explains the steeper evolution for the Cplex step of R-ILP algorithm.

### 5.6. Energy consumption

The energy consumption induced by the algorithms is also evaluated through simulation using a mean arrival rate of the VNF-FG request set to $\lambda = 5$ since the algorithms accept a maximum number of requests for this system load. This enables evaluation of energy consumption when a maximum number of requests are hosted (placed). The consumed energy is assessed by collecting the number of physical nodes that are switched on by each algorithm to serve the requests and the amount of power that is consumed for these requests. These two key performance indicators reflect directly the energy efficiency of the algorithms and enable their comparison.

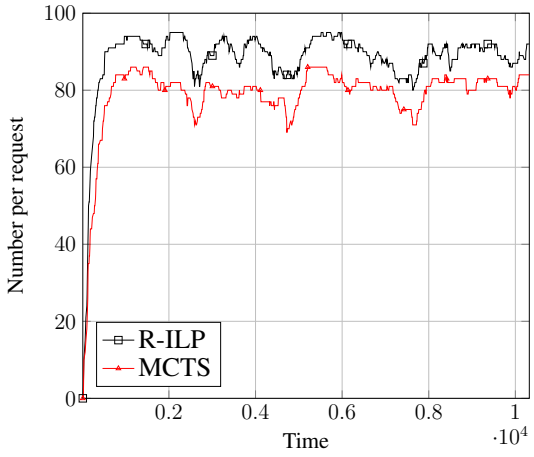


Figure 15: Number of switched ON servers

Figure 15 reports the total number of physical machines that are switched ON to serve the clients' requests and indicates that MCTS switches fewer physical nodes than the R-ILP algorithm.This observation can be misleading if the number of accepted requests is not an-

alyzed jointly with this energy metric. MCTS accepts only 807 requests out of the expressed 1000 requests in the simulation while the R-ILP accepts 968 requests. If the analysis is conducted jointly, for instance by normalizing the number of switched servers by the number of accepted requests, we obtain a more reliable metric or indicator as reported in Figure 16 that reveals that in fact the MCTS algorithm uses on average more physical machines to host each request. The R-ILP uses fewer physical machines to host the requests while serving after all more users.

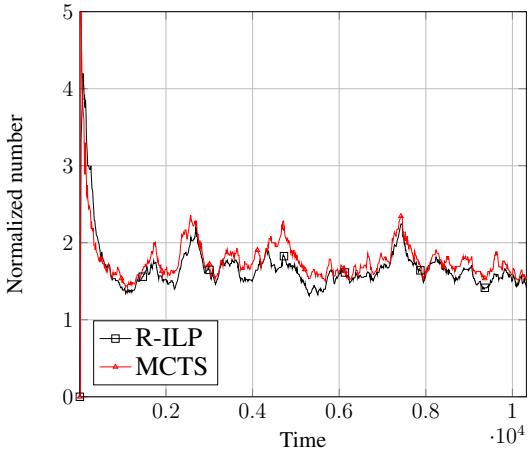

Figure 16: Normalized number of switched ON servers

Since the number of switched ON machines can not precisely evaluate the consumed energy in the entire NFV-I or data center, we extend the analysis by assessing the power consumed by physical machines for each algorithm and this same consumed power normalized by the number of accepted requests. The energy consumed by the NFV-I for each algorithm, due to the amount of used resources and the number of solicited hosts, is depicted in Figure 17 that quantitatively analyzes the evolution of the total consumed energy $P_t(G)$ in the data center at each instant $t$ based on equation 2. We fix $P^i(w) = 0$ and $P^M(w) = 50\ \forall w \in V(G)$ in the equation 2 for this evaluation. Even if the R-ILP algorithm accepts and serves more clients compared with the MCTS strategy, it consumes less energy thanks to its objective function, that computes the VNF-FG placement and chaining that leads to minimum global energy $P_t(G)$ consumption. To highlight the superior performance of the R-ILP across the entire infrastructure, Figure 18 depicts the consumed energy normalized by the number of accepted requests. This corresponds to the ratio of the "total consumed energy $P_t(G)$ at time $t$ in the NFV-I" to "the number of currently hosted VNF-FG requests". This key performance indicator (ratio or normalized consumed power) reports the energy consumed on average for each accepted request. The R-ILP is clearly seen to outperform the MCTS with gaps in consumed energy reaching up to 15 watts for the simulated scenarios (40 watts for the R-ILP versus 55 watts for MCTS).

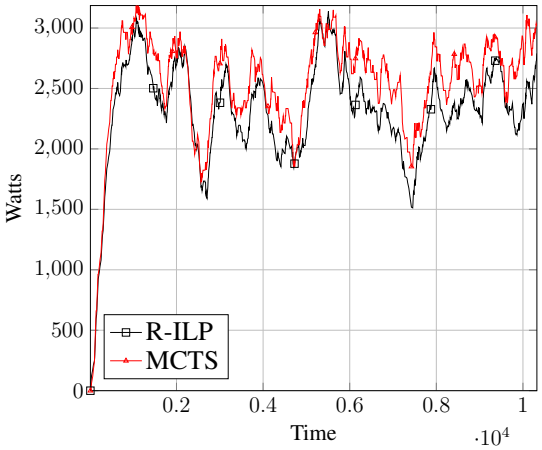

Figure 17: Energy consumption

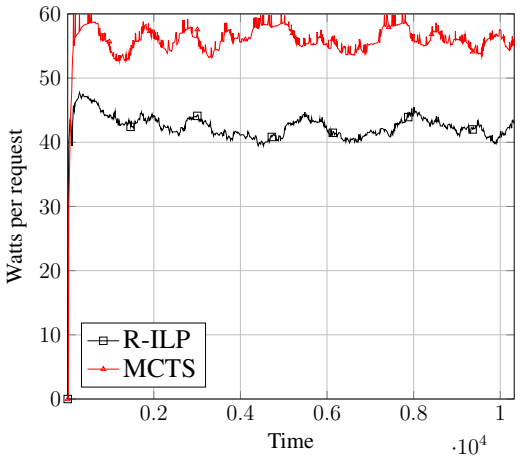

Figure 18: Normalized energy consumption

## 6. Conclusion

This paper proposes two embedding algorithms for the VNF-FG placement and chaining problem in an online and a batch mode. The online strategy exploits an integer linear program derived from a mathematical model of the VNF-FG placement problem taking into account the sharing of VNFs across tenants (or requests). The proposed ILP exploits the sharing capability offered by the NFV paradigm to optimize further physical resource usage and power consumption. The introduced batch mode algorithm, to enhance further overall performance, uses our R-ILP algorithm to process a group of requests, queued in a batch window, by serving in

priority requests that generate higher revenues for the providers while aiming at maximizing the number of served (i.e., placed, chained, hosted) requests.

The batch mode gives the opportunity to the service providers to select from a set of requests the most relevant ones to be served in priority based on their objectives. These two proposals were evaluated using extensive simulations and a relevant and fair comparison with a VNF sharing based algorithm called MCTS Competitor. The obtained results confirm the ability of our proposed ILP based algorithms to scale with problem size since they select a limited number of candidate hosts to reduce and control complexity. The proposed algorithms are shown to outperform in energy efficiency, rejection rate and achieved revenues the current state of the art through a comparison with a reference algorithm (MCTS).

The batch mode proposed in this work deserves additional attention by exploring to what extent it can use the ILP to treat groups of requests jointly without jeopardizing acceptance rate and quality of experience. Our future work will pursue the investigation and modeling to find the best possible trade offs.

## References


[1] O. Soualah, M. Mechtri, C. Ghribi, D. Zeghlache, A green vnfs placement and chaining algorithm, in: 2018 IEEE/IFIP Network Operations and Management Symposium, NOMS, Taipei, Taiwan, April 23-27, 2018.

[2] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, On orchestrating virtual network functions, in: Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), CNSM '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 50–56. doi:10.1109/CNSM.2015.7367338.

[3] A. Marotta, A. Kassler, A power efficient and robust virtual network functions placement problem, in: 28th International Teletraffic Congress, ITC 2016, Würzburg, Germany, September 12-16, 2016, 2016, pp. 331–339. doi:10.1109/ITC-28.2016.151.

[4] O. Soualah, M. Mechtri, C. Ghribi, D. Zeghlache, Energy efficient algorithm for VNF placement and chaining, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, pp. 579–588.

[5] O. Soualah, M. Mechtri, C. Ghribi, D. Zeghlache, An efficient algorithm for virtual network function placement and chaining, in: 14th IEEE Annual Consumer Communications & Networking Conference, CCNC 2017, Las Vegas, NV, USA, January 8-11, 2017, pp. 647–652.

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

[7] S. Mehraghdam, M. Keller, H. Karl, Specifying and placing chains of virtual network functions, in: Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on, 2014, pp. 7–13. doi:10.1109/CloudNet.2014.6968961.

[8] H. Moens, F. De Turck, VNF-P: A model for efficient placement of virtualized network functions, in: Network and Service Management (CNSM), 2014, pp. 418–423. doi:10.1109/CNSM.2014.7014205.

[9] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, D. Soldani, A novel approach to virtual networks embedding for SDN management and orchestration, in: 2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014, 2014, pp. 1–7. doi:10.1109/NOMS.2014.6838244.

[10] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, Near optimal placement of virtual network functions, in: 2015 IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1346–1354.

[11] B. Addis, D. Belabed, M. Bouet, S. Secci, Virtual network functions placement and routing optimization, in: 2015 IEEE 4th International Conference on Cloud Networking (CloudNet), 2015, pp. 171–177. doi:10.1109/CloudNet.2015.7335301.

[12] X. Li, C. Qian, The virtual network function placement problem, in: 2015 IEEE Conference on Computer Communications Workshops, INFOCOM Workshops, Hong Kong, China, April 26 - May 1, 2015, 2015, pp. 69–70. doi:10.1109/INFCOMW.2015.7179347.

[13] R. Bruschi, A. Carrega, F. Davoli, A game for energy-aware allocation of virtualized network functions, J. Electrical and COMPUTER Engineering 2016 (2016) 4067186:1–4067186:10. doi:10.1155/2016/4067186.

[14] G. D. Forney, The viterbi algorithm, Proceedings of the IEEE 61 (3) (1973) 268–278.

[15] M. Mechtri, C. Ghribi, D. Zeghlache, Vnf placement and chaining in distributed cloud, in: the 9th IEEE International Conference on Cloud Computing, June 27 - July 2, 2016, San Francisco, USA.

[16] C. Ghribi, M. Mechtri, D. Zeghlache, A dynamic programming algorithm for joint vnf placement and chaining, in: CoNEXT 2016, CAN, December 12 - 15, 2016, Irvine, USA, ACM, 2016.

[17] O. Soualah, M. Mechtri, C. Ghribi, D. Zeghlache, A link failure recovery algorithm for virtual network function chaining, in: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, May 8-12, 2017, pp. 213–221.

[18] M. T. Beck, J. F. Botero, K. Samelin, Resilient allocation of service function chains, in: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, November 7-10, 2016, 2016, pp. 128–133. doi:10.1109/NFV-SDN.2016.7919487.

[19] J. Fan, C. Guan, Y. Zhao, C. Qiao, Availability-aware mapping of service function chains, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057153.

[20] S. Khebbache, M. Hadji, D. Zeghlache, Virtualized network functions chaining and routing algorithms, Computer Networks 114 (2017) 95 – 110. doi:https://doi.org/10.1016/j.comnet.2017.01.008.

[21] A. Schrijver, Theory of linear and integer programming, John Wiley & Sons, Inc.

[22] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, V. Sekar, Stratos: A network-aware orchestration layer for middleboxes in the cloud, CoRR abs/1305.0209.

[23] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, P. Demeester, Network service chaining with optimized network function embedding supporting service decompositions, Computer Networks 93, Part 3 (2015) 492 – 505, cloud Networking and Communications {II}. doi:http://dx.doi.org/10.1016/j.comnet.2015.09.035.

[24] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, L. Gaspary, Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions, in: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, 2015, pp. 98–106. doi:10.1109/INM.2015.

7140281.

[25] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, T. Wood, Virtual function placement and traffic steering in flexible and dynamic software defined networks, in: 2015 IEEE International Workshop on Local and Metropolitan Area Networks, LANMAN 2015, Beijing, China, April 22-24, 2015, 2015, pp. 1–6. doi:10.1109/LANMAN.2015.7114738.

[26] K. Yang, H. Zhang, P. Hong, Energy-aware service function placement for service function chaining in data centers, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6. doi:10.1109/GLOCOM.2016.7841805.

[27] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, H. A. Chan, Optimal virtual network function placement in multi-cloud service function chaining architecture, Comput. Commun. 102 (C) (2017) 1–16. doi:10.1016/j.comcom.2017.02.011.

[28] ETSI GS NFV 001: Network Functions Virtualisation (NFV); Use Cases.

[29] ETSI GS NFV 003: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV (2014).

[30] M. Pedram, I. Hwang, Power and performance modeling in a virtualized server system, in: 39th International Conference on Parallel Processing, ICPP Workshops 2010, San Diego, California, USA, 13-16 September 2010, 2010, pp. 520–526. doi:10.1109/ICPPW.2010.76.

[31] IBM ILOG CPLEX Optimization Studio. [link].
URL https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[32] M. J. Todd, The many facets of linear programming, Math. Program. 91 (3) (2002) 417–436.

[33] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, 4th Edition, Springer Publishing Company, Incorporated, 2007.

[34] Y. Zhu, M. Ammar, Algorithms for assigning substrate network resources to virtual network components, IEEE INFOCOM (2006) 1–12.

[35] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, SIGCOMM Comput. Commun. Rev. 38 (2008) 17–29.

[36] Online and batch algorithms for vnfs placement and chaining.
URL https://github.com/MarouenMechtri/algorithms

[37] E. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, Proceedings of IEEE INFOCOM (1996) 594–602.

[38] S. Orlowski, M. Pióro, A. Tomaszewski, R. Wessäly, SNDlib 1.0–Survivable Network Design Library, in: Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium, 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.
URL http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz