Joint Admission Control and Embedding of SFC Requests in a Stochastic Environment for Maximizing Network Revenue

Daniela Cuesta
dcuesta@laas.fr
LAAS-CNRS, University of Toulouse, CNRS,
Toulouse, France, 31400

Olivier Brun
brun@laas.fr
LAAS-CNRS, University of Toulouse, CNRS,
Toulouse, France, 31400

Matthieu Jonckheere
matthieu.jonckheere@laas.fr
LAAS-CNRS, University of Toulouse, CNRS,
Toulouse, France, 31400

Balakrishna Prabhu
balakrishna.prabhu@laas.fr
LAAS-CNRS, University of Toulouse, CNRS,
Toulouse, France, 31400

Abstract

This paper studies the VNF Forwarding Graph embedding (VNF-FGE) problem in a stochastic setting with multiple classes of virtual network requests. Requests arrive as independent Poisson processes, occupy bandwidth and VNF-instance resources for exponential holding times, and yield class-dependent rewards when admitted and successfully deployed. The goal is to determine admission and routing rules that maximize the total expected reward in steady state. We propose scalable heuristics for two settings: *full-information*, where the network topology, capacities, and class requirements are known; and *partial-information*, where only class labels, partial congestion information, and outcomes are observed. For the full-information case, we use a combination of projected gradient and fixed-point approximations; and for the partial-information case, we use a reinforcement-learning approach (SAGE) to learn routing probabilities from limited observations. Numerical experiments illustrate the performance of the methods.

Keywords

Virtual network functions, loss networks, reduced-load approximation, projected gradient, reinforcement learning.

1 Introduction

Network Function Virtualization (NFV) replaces expensive, specialized hardware with flexible software versions called virtual network functions (VNF) that run on standard servers. This lowers costs and enable isolated network slices, tailored for specific applications. A virtual network (VN) is defined as a set of VNFs through which data packets should flow. The virtual links established between these network functions are described by a VNF Forwarding Graph (VNF-FG). The simplest example is that of a Service Function Chain (SFC) in which data packets emitted by a source node traverse a sequence of network functions before reaching their destination. As shown in Figure 1, it is possible to define multiple SFC in the same VNF-FG, each corresponding to a particular network processing path in the VNF-FG. A classifier is in charge of dynamically steering traffic flows to the appropriate SFC.

The VNF Forwarding Graph Embedding (VNF-FGE) problem is a key problem in that context [6]. It amounts to mapping VN requests to a given physical substrate network in order to optimize a certain performance objective.

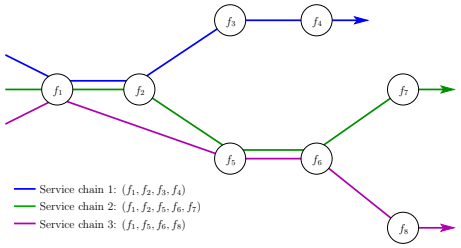

Figure 1: An example of VNF-FG.

This problem has been studied in the deterministic offline setting, which is an extension of the Virtual Network Embedding problem [5, 9], known to be NP-hard [1]. The problem has also been studied as an online optimization problem in [2, 10, 11, 13].

To the best of our knowledge, the present paper is the first to address the VNF-FGE problem in a stochastic setting. We assume that there are several classes of VN requests, each corresponding to an SFC with known resource requirements. Requests arrive randomly over time according to Poisson processes and, if admitted and deployed, they remain for an exponential duration and generate a reward. The problem at hand is thus to determine admission control rules and an embedding strategy that jointly optimize the total expected reward. Since the optimal policy is out-of-reach for large networks, we focus on state-independent policies, that is, policies in which routing decisions do not depend on the state of the system.

On one hand, we study a full-information setting where the decision-maker knows the topology, capacity constraints, the VNF-FG, and the resource requirements of each class. While one can optimize routing probabilities via steady-state blocking and gradient methods, exact evaluation becomes impractical in large networks due to state-space explosion, motivating scalable fixed-point approximations of the blocking probabilities.

On the other hand, we study partially observed regimes (black-box operation, multiple domains, or time-varying capacities) where the controller observes only the request class, limited congestion information, and realized outcomes, and must learn routing probabilities. We tackle this model-free setting using SAGE [4], a policy-gradient method leveraging product-form structure.

The paper is organized as follows. Section 2 states the problem. Section 3 presents exact and approximate gradient-based methods for the full-information case, and Section 4 presents the reinforcement learning approach. Numerical results are presented in Section 5, and conclusions in Section 6.

INOC '26, Liège (Belgium)

10.48786/inoc.2026.18

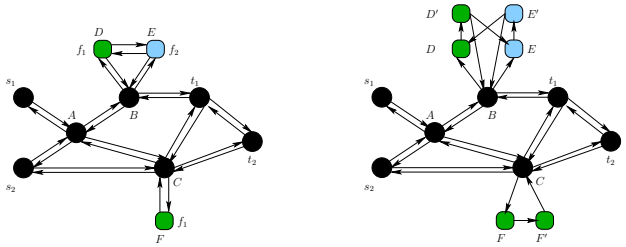

Figure 2: Example network (left) and its auxiliary network with duplicated function nodes and virtual links (right).

2 Problem Description

We are given a physical substrate network in which some nodes have the required compute and storage resources to host VNF; we call these nodes *function nodes*. A function node corresponds to a logical entity representing reserved computing and storage capacity in a data center for executing one or more instances of a particular VNF, and its capacity is the number of available VNF instances. Multiple function nodes may be colocated in the same data center; we model them as fully connected with infinite-capacity links.

We assume that there are multiple classes of VN requests. Each class is characterized by a source, a destination node, and a sequence of VNFs through which the corresponding traffic must flow, called the *Service Function Chain* (SFC).

The left panel of Figure 2 shows a simple example of the setting considered in the paper. There are two VNFs: f_1 and f_2, and three function nodes: D, E and F, hosting instances of VNFs f_1, f_2 and f_1, respectively. Function nodes D and E are colocated in the same data center which is connected to node B. Note also that instances of VNF f_1 may be reserved either at node D or at node F. Class 1 requests correspond to VNs established between nodes s_1 and t_1, whose traffic must flow through VNF f_1 first and then f_2. Class 2 requests correspond to VNs between s_2 and t_2, and the corresponding traffic must flow through VNF f_2. In this example, both classes allow multiple resource allocations.

The problem at hand is to decide whether an incoming VN request should be admitted and, if admitted, to choose a single path through the network such that the corresponding function nodes are visited in the prescribed order. Since capacities are associated both with transmission links (in terms of bandwidth units) and to function nodes (in terms of number of VNF instances), we use a standard graph transformation that yields a model with link capacities only: each function node is duplicated and an artificial edge is introduced between the two copies, whose capacity represents the number of available VNF instances. Figure 2 shows the original network (left) and the resulting auxiliary network (right), with duplicated function nodes and virtual links.

We represent the auxiliary network as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. The capacity of edge j is C_j circuits, representing either bandwidth units available on a "normal" link or VNF instances available on an artificial link. We use the term "circuit" for both.

There are K classes of VN requests. Requests of class k arrive according to a Poisson process at rate λ_k and, if admitted and deployed, they remain in the network for an exponential time with mean $1/\mu_k$. We denote by $\rho_k = \lambda_k/\mu_k$ the offered traffic of class k. The holding times are assumed independent of earlier arrivals, of previous holding times, and of the route used to establish the request. The resources allocated to a VN request are reserved for this request as long as the request is present in the system.

We let $\mathcal{R}_k$ be a set of feasible routes for class-k requests, and denote by $\mathcal{R}$ the set of all routes. To simplify notation, we assume that they are numbered from 1 to R, so that the index of a route uniquely identifies both the route itself and the request class. We define $a_{j,r}$ as the number of circuits reserved on link j for a request routed on route r. When a class-k request arrives, it is assigned to route $r \in \mathcal{R}_k$ with a fixed routing probability, and it is deployed on r only if there are at least $a_{j,r}$ free circuits on each link $j \in r$; otherwise, it is blocked. Defining $n_r(t)$ as the number of requests present on route r at time t, the state of the system at time t is described by the vector $\mathbf{n}(t) = (n_r(t))_{r\in\mathcal{R}}$, which belongs to the set of all feasible states $\mathcal{S}(\mathbf{C})$, where

$$\mathcal{S}(\mathbf{C}) = \left\{\mathbf{n} \in \mathbb{Z}_+^R \ : \ A\,\mathbf{n} \le \mathbf{C}\right\}.$$

and where $A = [a_{j,r}]_{j\in E, r\in\mathcal{R}}$ and $\mathbf{C} = (C_j)_{j\in E}$.

A request arriving at time t and assigned to route r is therefore deployed on r only if

$$a_{j,r} \le C_j - \sum_{p\in\mathcal{R}} a_{j,p} n_p(t), \quad \text{for all } j \in r,$$

or equivalently, if $\mathbf{n}(t) \in \mathcal{S}(\mathbf{C} - A\mathbf{e}_\mathbf{r})$. If the request is admitted and deployed, it generates a revenue w_r (which depends only on the class of the request, that is, $w_r = w_k$ for $r \in \mathcal{R}_k$) and simultaneously holds $a_{j,r}$ circuits on each link $j \in r$ for the duration of the request. Our goal is to determine an admission control and dynamic routing policy to maximize the total expected reward in steady-state.

We restrict ourselves to state-independent policies $\mathbf{x}$, that is, routing decisions do not depend on $\mathbf{n}$. For such policies, let

$$x(r) = \mathbb{P}(\text{Admit an arriving VN request on route } r),$$

with $\sum_{r\in\mathcal{R}_k} x(r) \le 1$ for each class k. Then class-k VN requests arrive on route $r \in \mathcal{R}_k$ at rate $\lambda_r = \lambda_k x(r)$ and the offered traffic is $y_r = \rho_k x(r)$. Note that the inequality $\sum_{r\in\mathcal{R}_k} x(r) \le 1$ may be strict to reserve capacity for more profitable requests.

Under any state-independent policy $\mathbf{x}$, the stochastic process $\{\mathbf{n}(t)\}_{t\ge 0}$ is a continuous-time Markov chain over the state space $\mathcal{S}(\mathbf{C})$, with a unique stationary distribution π. The total network revenue is

$$W(\mathbf{y}) = \sum_{r\in\mathcal{R}} w_r \mathbb{E}\left[n_r(t)\right],$$

where the expectation is taken with respect to π.

As discussed in [8], π has the product-form

$$\pi(\mathbf{n}) = \frac{1}{G(\mathbf{C})} \prod_{r\in\mathcal{R}} \frac{y_r^{n_r}}{n_r!}, \quad \text{for all } \mathbf{n} \in \mathcal{S}(\mathbf{C}), \tag{1}$$

where the normalizing constant $G(\mathbf{C})$ is given by

$$G(\mathbf{C}) = \sum_{\mathbf{n}\in\mathcal{S}(\mathbf{C})} \prod_{r\in\mathcal{R}} \frac{y_r^{n_r}}{n_r!}. \tag{2}$$

The blocking probability of route r, denoted by $L_r(\mathbf{C})$, satisfies

$$1 - L_r(\mathbf{C}) = \sum_{\mathbf{n}\in\mathcal{S}(\mathbf{C}-A\mathbf{e}_\mathbf{r})} \pi(\mathbf{n}) = \frac{G(\mathbf{C} - A\mathbf{e}_\mathbf{r})}{G(\mathbf{C})}, \tag{3}$$

and $\mathbb{E}\left[n_r(t)\right] = y_r(1 - L_r(\mathbf{C}))$. Therefore

$$W(\mathbf{y}) = \sum_{r\in\mathcal{R}} w_r y_r (1 - L_r(\mathbf{C})). \tag{4}$$

To convert the inequalities constraints $\sum_{r\in\mathcal{R}_k} y_r \le \rho_k$ for all $k = 1,\dots,K$ into equalities, for each class k we introduce an artificial route with only one link, say route 0, and set $\mathcal{R}_k^* = \mathcal{R}_k \cup \{0\}$. A rejected class k request is modeled as being routed on route 0 for this class, with $C_0 = \infty$, $L_0(\mathbf{C}) = 0$ and $w_0 = 0$. Denoting by $\mathcal{R}^*$ the set of all routes including these fictitious routes (one per class), the optimization problem can be formulated as follows:

$$\begin{aligned} \text{Maximize } & W(\mathbf{y}) = \sum_{r\in\mathcal{R}^*} w_r y_r (1 - L_r(\mathbf{C})), \\ \text{s.t.} & \\ & \sum_{r\in\mathcal{R}_k^*} y_r = \rho_k, \quad k = 1,\dots,K, \\ & y_r \ge 0 \quad \forall r \in \mathcal{R}^*. \end{aligned} \tag{5}$$

3 Projected Gradient Method

In this section, we address the case in which the decision-maker has access to a full model of the system, including the network topology, capacity constraints, and the resource requirements of requests. This full-information setting enables model-based optimization of state-independent routing policies: one can evaluate (exactly or approximately) the steady-state performance of a given policy and update routing probabilities using gradient-based methods.

3.1 Gradient Algorithm

To maximize W as expressed in (4), we use a gradient-based method. Under the product-form stationary distribution (1), the partial derivative of W with respect to y_r is given by

$$\frac{\partial W}{\partial y_r}(\mathbf{y}) = w_r(1 - L_r(\mathbf{C})) + \sum_{j\in\mathcal{R}} w_j y_j \left[(L_r(\mathbf{C}) - L_r(\mathbf{C} - A\mathbf{e}_j))\right]. \tag{6}$$

Algorithm 1 describes the procedure for computing the gradient ∇W of our objective function with respect to y_r, $r \in \mathcal{R}$.

Algorithm 1 Algorithm for computing ∇W

```
Compute G(C) with (2)
for j ∈ R do
    Compute G(C − Ae_j) with (2)
    1 − L_j(C) = G(C − Ae_j)/G(C)
    for r ∈ R do
        Compute G(C − A(e_j + e_r)) with (2)
        1 − L_r(C − Ae_j) = G(C − A(e_j + e_r))/G(C − Ae_j)
    end for
end for
for k = 1, ..., K do
    for r ∈ R_k do
        Compute ∂W/∂y_r with (6)
    end for
end for
```

The crucial steps in Algorithm 1 are the computations of the normalization constants $G(\mathbf{C})$, $G(\mathbf{C}-A\mathbf{e}_j)$ and $G(\mathbf{C}-A(\mathbf{e}_j+\mathbf{e}_r))$ for all $j, r \in \mathcal{R}$. This is feasible only for networks such that the size $S = |\mathcal{S}(\mathbf{C})|$ of the state space remains relatively small.

This problem can be solved using any feasible direction method. Starting from a feasible initial point $\mathbf{y}^{(0)} = \left(y_r^{(0)}\right)_{r\in\mathcal{R}^*}$, these methods update the current solution $\mathbf{y}^{(t)}$ at iteration t as follows

$$\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} + \alpha_t \, \Delta\mathbf{y}^{(t)}, \tag{7}$$

$\Delta\mathbf{y}^{(t)}$ is a feasible ascent direction at point $\mathbf{y}^{(t)}$ and α_t is the step size (chosen class by class).

3.2 Approximated gradient

For large networks, an alternative approach is to assume that links block independently, that is, that for all link $j \in E$ and all classes $k = 1,\dots,K$, there exists a parameter $B_{k,j} \in [0,1]$ such that

$$1 - L_r(\mathbf{C}) = \prod_{j\in r}(1 - B_{k,j}), \quad \text{for all } r \in \mathcal{R}_k. \tag{8}$$

The parameter $B_{k,j}$ represents the blocking probability of class-k requests on link j. This assumption is known to be asymptotically correct when offered traffic and link capacities are increased together at the same rate, in which case the parameters $B_{k,j}$ can be obtained as the solution of an optimization problem [8]. Under the above assumption, one can apply the well-known reduced-load approximation to determine these parameters [3]. Let $E_k(\nu_1,\dots,\nu_K;C)$ be a function which returns the class-k blocking probability on a single link with capacity C and offered traffic ν_i for class-i requests, $i = 1,\dots,K$. Under the so-called Knapsack approximation [3], the $B_{k,j}$ are obtained as the solutions of the non-linear equations

$$B_{k,j} = E_k\left(\nu_{1,j},\dots,\nu_{K,j};C_j\right), \quad \text{for all } j \in E, k = 1,\dots,K, \tag{9}$$

where

$$\nu_{k,j} = \sum_{r\in\mathcal{R}_k, j\in r} y_r \prod_{\ell\in r\setminus\{j\}} (1 - B_{k,\ell}).$$

The above system of non-linear equations is usually solved using a fixed-point algorithm. A similar but less accurate approximation scheme was also proposed by Kelly in [8]. It is known that there exists a solution to equations (9) but this solution is not necessarily unique (note that it is unique for the approximation scheme proposed in [8]). It is also shown in [3] and [8] that the approximation (8)-(9) converges to the correct values under a limiting regime in which offered traffic and link capacities are scaled-up together. The function $E_k(\nu_1,\dots,\nu_K;C)$ is usually implemented either as the Erlang-B function when $a_{j,r} = 1$ for all links j and routes r, or as the Kaufman-Roberts algorithm for multi-rate loss networks [7, 12]. Using (8)-(9), one can obtain an approximation of the end-to-end loss probabilities in large networks that are required in (6) for the computation of ∇W.

3.3 Projected Gradient (PG)

We apply projected gradient ascent to optimize the long-run reward W over state-independent routing policies.

For class k, we can compute the direction $\Delta\mathbf{y}$ at point $\mathbf{y}$ as

$$\Delta y_r = \frac{\partial W}{\partial y_r}(\mathbf{y}) - \frac{1}{|\mathcal{R}_k^*|}\sum_{i\in\mathcal{R}_k^*}\frac{\partial W}{\partial y_i}(\mathbf{y}), \quad \text{for all } r \in \mathcal{R}_k^*,$$

where $\frac{\partial W}{\partial y_0}(\mathbf{y})$ is set to 0. By construction $\sum_{r\in\mathcal{R}_k^*}\Delta y_r = 0$, so the update (7) preserves the constraint (5). However, at the boundary

the direction may point outside the feasible region (e.g., when $y_0 = 0$ and $\Delta y_0 < 0$). We therefore use two class-wise projections:

Case 1: if $y_0 > 0$, or if $y_0 = 0$ and $\sum_{i\in\mathcal{R}_k^*} \frac{\partial W}{\partial y_i}(\mathbf{y}) \le 0$, we project onto the hyperplane that preserves the sum over all routes (real routes and the extra route). This yields, for each $r \in \mathcal{R}_k^*$,

$$\Delta y_r = \frac{\partial W}{\partial y_r} - \frac{1}{|\mathcal{R}_k^*|} \sum_{i\in\mathcal{R}_k^*} \frac{\partial W}{\partial y_i}. \tag{10}$$

Case 2: If $y_0 = 0$ and $\sum_{i\in\mathcal{R}_k^*} \frac{\partial W}{\partial y_i}(\mathbf{y}) > 0$, the feasible directions must keep the class sum fixed on the boundary, so we project onto the hyperplane that preserves the sum over the real routes:

$$\Delta y_r = \frac{\partial W}{\partial y_r} - \frac{1}{|\mathcal{R}_k|} \sum_{i\in\mathcal{R}_k^*} \frac{\partial W}{\partial y_i} \text{ if } r \in \mathcal{R}_k \text{ and } \Delta y_0 = 0. \tag{11}$$

Note that specifying Δy_r over $\mathcal{R}^*$ via (10) or (11) is equivalent to first specifying Δy_r over $\mathcal{R}$, and then setting $y_0 = \rho_k - \sum_{i\in\mathcal{R}_k} y_i$.

Once $\Delta\mathbf{y}$ is computed, coordinates already at a boundary (0 or ρ) and pointing outward are frozen in the step computation.

After choosing the direction, to avoid steps that exit the feasible region, at each iteration t we set a maximum step size β_t and then choose α_t as the largest value in $(0, \beta_t]$ that preserves feasibility, i.e., all y_r remain nonnegative for all $r \in \mathcal{R}^*$. To choose β_t, we can simply set it to a fixed value.

4 Learning-Based Method

In many applications, the controller does not have access to a complete and explicit description of the network or the request constraints (e.g., topology, capacity limits, or detailed resource requirements), either because the system operates as a black box or because constraints are time-varying and only partially observable. In this setting, it is natural to approach the control problem in a reinforcement learning framework and to learn good routing probabilities and an estimate of the maximal achievable value of W directly from observed trajectories.

4.1 Score-aware gradient estimation (SAGE)

Classical policy-gradient methods (e.g., actor-critic) estimate policy gradients through the policy-gradient theorem [14], which typically involves a value or action-value function and therefore requires learning an additional critic.

In a standard average-reward reinforcement learning setting, a controller interacts with a stochastic system over repeated decision epochs. Under a parametrized stationary policy, the state evolves as a Markov chain: at each epoch the controller selects an action based on the current state, receives an immediate reward, and the system transitions to a new state. The goal is to choose the policy parameters that maximize the long-run average reward.

SAGE [4] exploits the analytical structure possibly present in the stationary distribution induced by the policy. When this distribution admits a product-form (equivalently, belongs to an exponential family), SAGE yields a specific gradient decomposition that combines a stationary-distribution term (estimable from sample covariance) with a policy-score term, avoiding value-function approximation.

The resulting SAGE-based policy-gradient algorithm follows the standard loop: first sample a trajectory (or a batch of decision epochs) under the current parameters, then form a sample-based estimate of the gradient using the SAGE decomposition, and finally update the parameters via stochastic gradient ascent.

Empirically, [4] reports faster convergence and lower variance than actor-critic when applicable; as well as more robust behavior when not all policies are stable. This motivates using SAGE here, since our stochastic-network model provides exactly the product-form structure that SAGE is designed to exploit.

4.2 Markov Decision Process Formulation

We make routing decisions at request arrival times. Each arrival of class k is processed in two steps. First, the controller may reject the request immediately (this corresponds to assigning the remaining probability $1 - \sum_{r\in\mathcal{R}_k} x(r)$ to the fictitious route 0). Otherwise, the request is admitted and assigned to one of the candidate routes in its class-dependent set. After this assignment, the request is deployed on the network only if sufficient resources are available along the chosen route; if not, the request is blocked due to lack of capacity. We describe the Markov decision process as follows.

State. The state contains the current occupancies per route (in the real routes) $\mathbf{n}(t) = (n_r(t))_{r\in\mathcal{R}}$ and the class label of the arriving request $K(t)$.

Action. Given an arrival of class k, the action is to choose either one route $r \in \mathcal{R}_k$ or the reject option (choosing the fictitious route 0). Equivalently, given an arrival of class k, the action is to choose $r \in \mathcal{R}_k^* = \mathcal{R}_k \cup \{0\}$. Notice that the only information of the state that we use to decide the action is the class label, because the routing probabilities we consider are independent of the occupancy $\mathbf{n}$.

Reward. Each class k has an associated reward $w_k > 0$. The reward equals w_k if the request is deployed in the network (that is, if it is admitted and sufficient capacity is available on the chosen route), and it equals 0 if the request is rejected immediately or blocked due to lack of resources on the chosen route.

Dynamics. Arrivals occur by class according to Poisson processes. If an arrival of class k is rejected, the occupancies do not change. If it is assigned to route $r \in \mathcal{R}_k$ and sufficient capacity is available, the occupancies increase according to the resource requirements of that route. If capacity is insufficient, the request is blocked and the occupancies do not increase. Requests that are deployed remain in the network for an exponentially distributed time and then depart, releasing their resources; thus the occupancies evolve through both arrivals and departures. The next decision epoch is the next arrival time; the data used for gradient estimation is collected over a batch of such arrival epochs.

This description, together with the product-form (1), provides the required structure to apply SAGE.

5 Numerical Results

We consider a simple example network with routers A, B and C, and function nodes D, E and F. Node D provides instances of VNF f_1, whereas E and F provide instances of VNF f_2. Nodes D and E are colocated in a data center connected to router A, whereas function node F is in another data center connected to router C. Links are symmetric with the same capacity in both directions, and dotted links have infinite capacity in both directions.

We use the auxiliary network in Figure 3. The normal links (A, B), (A, C) and (B, C) have capacities 16, 20 and 12 respectively; the virtual links (D, D'), (E, E') and (F, F') have capacities 20, 12 and 8 respectively. All other links have infinite capacity.

The three request classes are summarized in Table 1, which lists, for each class, the source and destination nodes, admissible

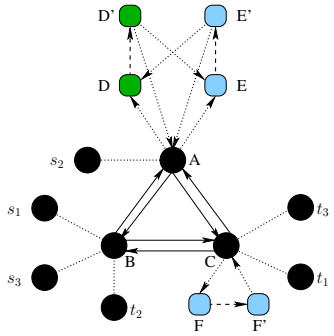

Figure 3: Auxiliary network with three virtual links.

Table 1: Characteristics of the traffic classes.

ρ	src	dst	w_k	f_1	f_2	BW	routes
2.0	s_1	t_1	1	2	0	4	r_1, r_2
2.4	s_2	t_2	1	0	1	1	r_3, r_4
2.128	s_3	t_3	10	1	2	2	r_5, r_6

routes, VNF instance requirements, offered load, and bandwidth units per transmission link. Routes from r_1 to r_6 are listed below.

$r_1 = \{s_1, B, A, D, D', A, C, t_1\}$ $r_2 = \{s_1, B, C, A, D, D', A, C, t_1\}$
$r_3 = \{s_2, A, E, E', A, B, t_2\}$ $r_4 = \{s_2, A, C, F, F', C, B, t_2\}$
$r_5 = \{s_3, B, A, D, D', E, E', A, C, t_3\}$ $r_6 = \{s_3, B, A, D, D', A, C, F, F', C, t_3\}$

We maximize the network revenue W in (4) using PG with gradient given by (6), and SAGE. We consider two PG variants: PG-exact, which computes blocking probabilities via the product-form solution (1)-(2), and PG-approx, which uses the Knapsack approximation [3] described in (8) and (9), avoiding state enumeration. We run both PG variants with fixed step size 0.1 and plot the routing trajectories and the corresponding objective value (denoted W for PG-exact and W_{approx} for PG-approx). For SAGE, we run 2000 updates with batch size 50 (i.e., a total of 10^5 arrivals), and step size $1/\sqrt{t}$; at 51 checkpoints (every 400 updates), we freeze the routing vector and estimate W by Monte Carlo (MC) averaging 10 independent simulations of 1000 arrivals each. We also compute the exact W at the same routing vectors.

We set $\mathcal{R} = \{1, \ldots, 6\}$ and let $\mathbf{x} = (x(1), \ldots, x(6))$, where $x(i) = x(r_i)$, $i \in \mathcal{R}$. The coordinates are grouped by class as $(x(1), x(2))$, $(x(3), x(4))$ and $(x(5), x(6))$. For class 1, the rejection probability is $1 - x(1) - x(2)$, and analogously for the other classes.

We start from $\mathbf{x} = (0.75, 0.25, 0.167, 0.833, 0.235, 0.235)$, where exact $W \approx 13.202$; this is suboptimal, since most class-3 traffic (whose weight is 10) is rejected.

PG-exact and PG-approx end at the following routing vector $(0, 1, 1, 0, 0.54, 0.46)$ (difference 0.2 in the last two coordinates), where exact $W \approx 24.402$ and $W_{\text{approx}} \approx 23.998$, and show similar behavior from other random initializations. Figure 4 plots the objective value as a function of the number of updates (exact W for PG exact, and W_{approx} for PG approx), and Figure 5 plots the routing probabilities.

Figure 6 shows the SAGE results. Over the last 10 checkpoints (updates 1640-2000) the MC values have mean 23.67 (range 23.28-24.15) and the exact-W values have mean 22.80 (range 22.79-22.83). We also ran the same initialization for 20000 updates: over the last 10 checkpoints (updates 16400-20000), the MC values have mean 24.36 and the exact-W values have mean 23.14 (range 23.11-23.17), suggesting that most gains occur early, while longer runs mainly reduce variance and improve stability. With batch size

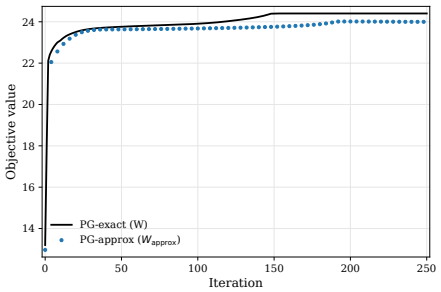

Figure 4: PG-exact and PG-approx routing trajectories (0–250 iterations); exact W in black, approximate W in blue markers.

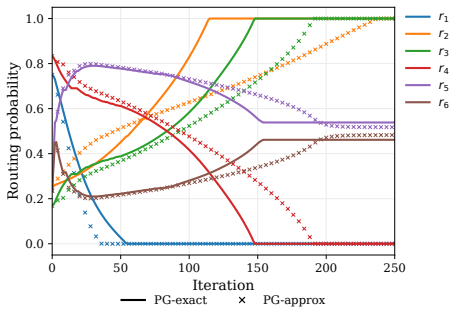

Figure 5: Time evolution of routing probabilities for PG-exact (solid) and PG-approx (crosses), same color per route.

10, noise increases and stability decreases, but the objective still reaches values bigger than 22.3 after 250 updates (2500 arrivals). Other initializations show the same qualitative behavior.

We also consider larger and structurally more complex network-demand instances. In one of the examples considered, the number of feasible states is more than 15 million, with 68 directed links, 5 VNFs, 20 classes and 3 admissible routes per class. PG-exact is infeasible at this scale. By contrast, SAGE remains feasible because it relies on trajectory data. Starting from a random initialization, SAGE increases the MC estimate of W from 42.84 to 63.53 (average of 10 runs, each with 10000 arrivals). For PG-approx, we applied a damped fixed-point iteration to avoid oscillation. With damping, the approximate objective increased from 38.16 to 63.70. Differences between the SAGE and the PG-approx values are expected because MC is an estimator and the reduced-load approximation can be inaccurate when the link-independence assumption is violated.

6 Conclusions

We have addressed the VNF-FGE problem in a stochastic setting. To determine the admission control policy and the SFC embedding policy which jointly maximize the total network revenue (among the class of state-independent policies), we have considered two schemes: projected gradient, efficient when the network topology is known, and reinforcement learning (SAGE), suitable when the topology is only partially known. In the small

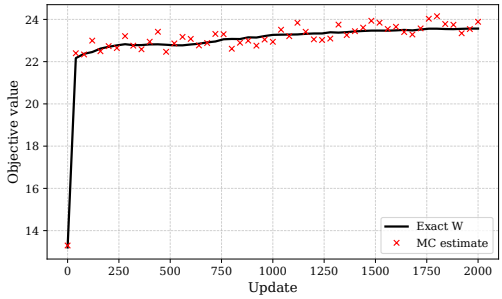

Figure 6: SAGE performance over 2000 updates (batch size 50): exact objective value (solid black) and MC estimate (red crosses).

network example, PG-approx reaches essentially the same maximizer as the exact method, but much faster, whereas SAGE improves substantially from the initial policy but is less accurate in this instance. In the larger instance, exact evaluation is infeasible, but both methods show clear gains, with PG-approx improving the objective with fewer iterations. To stabilize the reduced-load computation in PG-approx, we used a damped fixed-point iteration. Although the reduced-load approximation can be inaccurate when the link-independence assumption is violated, SAGE can still be applied since it only uses trajectory data to update the routing parameters.

Acknowledgments

This work was supported by the ANR under the France 2030 program, grant "NF-NAI: ANR-22-PEFT-0003".

References

[1] Edoardo Amaldi, Stefano Coniglio, Arie M. C. A. Koster, and Martin Tieves. 2016. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics* 52 (2016), 213–220. doi:10.1016/j.endm.2016.03.028

[2] Francisco Carpio, Wolfgang Bziuk, and Admela Jukan. 2017. Replication of Virtual Network Functions: Optimizing Link Utilization and Resource Costs. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, Opatija, Croatia, 521–526. doi:10.23919/MIPRO.2017.7973481

[3] Shun-Ping Chung and Keith W. Ross. 1993. Reduced load approximations for multirate loss networks. *IEEE Transactions on Communications* 41, 8 (1993), 1222–1231. doi:10.1109/26.231966

[4] Céline Comte, Matthieu Jonckheere, Jaron Sanders, and Albert Senen-Cerda. 2025. Score-Aware Policy-Gradient and Performance Guarantees using Local Lyapunov Stability. *Journal of Machine Learning Research* 26, 132 (2025), 1–74. HAL:hal-04329790v1 https://www.jmlr.org/papers/v26/24-1009.html

[5] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. 2013. Virtual Network Embedding: A Survey. *IEEE Communications Surveys & Tutorials* 15, 4 (2013), 1888–1906. doi:10.1109/SURV.2013.013013.00155

[6] Juliver Gil Herrera and Juan Felipe Botero. 2016. Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management* 13, 3 (2016), 518–532. doi:10.1109/TNSM.2016.2598420

[7] Joseph S. Kaufman. 1981. Blocking in a shared resource environment. *IEEE Transactions on Communications* 29, 10 (1981), 1474–1481. doi:10.1109/TCOM.1981.1094894

[8] Frank P. Kelly. 1986. Blocking probabilities in large circuit-switched networks. *Advances in Applied Probability* 18, 2 (June 1986), 473–505. doi:10.2307/1427309

[9] Abdelquoddouss Laghrissi and Tarik Taleb. 2019. A Survey on the Placement of Virtual Resources and Virtual Network Functions. *IEEE Communications Surveys & Tutorials* 21, 2 (2019), 1409–1434. doi:10.1109/COMST.2018.2884835

[10] Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. 2016. VNF Placement and Chaining in Distributed Cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, San Francisco, CA, USA, 376–383. doi:10.1109/CLOUD.2016.0057

[11] Tran Anh Quang Pham, Abbas Bradai, Kamal Deep Singh, Gauthier Picard, and Roberto Riggio. 2019. Single and Multi-Domain Adaptive Allocation Algorithms for VNF Forwarding Graph Embedding. *IEEE Transactions on Network and Service Management* 16, 1 (2019), 98–112. doi:10.1109/TNSM.2018.2876623

[12] James W. Roberts. 1981. A Service System with Heterogeneous User Requirements – Application to Multi-Service Telecommunication Systems. In *Performance of Data Communication Systems and their Applications*, Guy Pujolle (Ed.). North-Holland, Amsterdam, 423–431.

[13] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. 2018. A Green VNF-FG Embedding Algorithm. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, Montreal, QC, Canada, 141–149. doi:10.1109/NETSOFT.2018.8460013

[14] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). The MIT Press, Cambridge, MA, USA.