A Constrained Shortest Path Scheme for Virtual Network Service Management

Dmitrii Chemodanov*, Flavio Esposito†*, Prasad Calyam*, Andrei Sukhov‡

*University of Missouri, USA; †Saint Louis University, USA; ‡Samara National Research University, Russia;
Email: *dycbt4@mail.missouri.edu, flavio.esposito@slu.edu, calyamp@missouri.edu, amskh@yandex.ru*

***Abstract*—Virtual network services that span multiple data centers are important to support emerging data-intensive applications in fields such as bioinformatics and retail analytics. Successful virtual network service composition and maintenance requires flexible and scalable 'constrained shortest path management' both in the management plane for virtual network embedding (VNE) or network function virtualization service chaining (NFV-SC), as well as in the data plane for traffic engineering (TE). In this paper, we show analytically and empirically that leveraging constrained shortest paths within recent VNE, NFV-SC and TE algorithms can lead to network utilization gains (of up to 50%) and higher energy efficiency. The management of complex VNE, NFV-SC and TE algorithms can be, however, intractable for large scale substrate networks due to the NP-hardness of the constrained shortest path problem. To address such scalability challenges, we propose a novel, exact constrained shortest path algorithm viz., 'Neighborhoods Method' (NM). Our NM uses novel search space reduction techniques and has a theoretical quadratic speed-up making it practically faster (by an order of magnitude) than recent branch-and-bound exhaustive search solutions. Finally, we detail our NM-based SDN controller implementation in a real-world testbed to further validate practical NM benefits for virtual network services.**

***Index Terms*—Constrained Shortest Path, Virtual Network Embedding, NFV Service Chaining, Traffic Engineering.**

I. Introduction

THE advent of network virtualization has enabled new business models allowing infrastructure providers to share or lease their physical networks that span multiple data centers. Network virtualization is being increasingly adopted to support data-intensive applications within enterprises ($e.g.$, retail analytics) and academia ($e.g.$, bioinformatics and high energy physics) over wide-area federated infrastructures such as the VMware Cloud [1] and the Global Environment for Network Innovations (GENI) [2]. A major challenge for infrastructure providers is to offer virtual network services that meet the application Service Level Objective (SLO) demands on shared (constrained) physical networks. Examples of SLO demands can refer to technical constraints such as bandwidth, high reliability, or low latency.

In order to compose and maintain virtual network services, infrastructure providers need to run management protocols (within the 'management plane') such as $e.g.$, Virtual Network Embedding (VNE) to satisfy users' virtual network requests. VNE is the NP-hard graph matching problem of mapping a constrained virtual network on top of a shared physical infrastructure [3], [4]. Another management protocol example pertains to Network Function Virtualization service chaining (NFV-SC) [5], [6], in which a network manager is required to place Virtual Network Functions ($e.g.$, firewalls, load balancers, etc) and setup a path across corresponding software-defined middleboxes to guarantee different high-level traffic constraints $i.e.$, policies. While virtual network services are operating, network managers need to also deploy, within the 'data plane', traffic engineering (TE) techniques to maintain profiles or improve network utilization [7].

Successful virtual network service composition and maintenance in both management plane and data plane mechanisms requires *scalable* and *flexible* 'constrained shortest path management' for the following reasons. The constrained shortest path problem is the NP-hard problem of finding the shortest path that satisfies an arbitrary number of end-to-end (or path) constraints [8], and its existing exact solutions are commonly based on branch-and-bound exhaustive search algorithms [8], [9], [10] or integer programming. These techniques have exponential complexity, and hence, limit *scalability* of the constrained shortest path management. Such scalability limitations are exacerbated by the complexity of VNE, NFV-SC and TE algorithms as well as the potentially large scale of substrate networks. Although some heuristics or approximation algorithms can find constrained shortest paths in polynomial time (at expense of optimality [11]), these algorithms support only a specific number of constraints, or a specific cost to optimize [12], [13], [14]. Thereby, they limit the *flexibility* of the constrained shortest path management.

By leveraging constrained shortest paths, we show how we can enhance network utilization and energy efficiency of VNE, NFV-SC and TE services by both analytical and empirical means. Reasons for the observed benefits are as follows: Firstly, by using a constrained shortest path, we can minimize a provider's cost associated with flows allocation subject to the application SLO constraints in TE [15], [16]. Such costs potentially hinder long-term infrastructure providers' revenue maximization; this can be understood using a simple example: a path comprising of two physical links to maintain a single flow has a higher allocation cost than an alternative solution that uses only one physical link. For example, if the flow SLO demand is 10 Mbps, a path composed by a single physical link would need to provision only 10 Mbps, while a path composed by two links would require 20 Mbps. Secondly, a more scalable and flexible constrained shortest path approach can be also beneficial for VNE and NFV-SC [17], [18], [19] when virtual links are subject to an arbitrary number of constraints such as bandwidth, latency and loss rate. Other types of constraints can also be imposed by optimization methods, such as the column generation approach, typically

This work has been partially supported by the National Science Foundation awards CNS-1647084, CNS-1647182, the Coulter Foundation Translational Partnership Program and by RFBR according to the research project 16-07-00218a and the public tasks of the Ministry of Education and Science of the Russian Federation (2.974.2017/4.6). Any opinions, findings or conclusions expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

used to speed-up integer programming [8], [17], [18]. This is because in the aforementioned cases, constrained shortest paths can be part of the optimal solution, $i.e.$, such paths can best improve the objective value under arbitrary constraints.

Our Contributions. To achieve the constrained shortest path benefits and address their management flexibility and scalability challenges in virtual network services, we propose a novel and exact constrained shortest path algorithm $viz.$, 'Neighborhoods Method' (NM). Our NM is based on a novel double-pass search space reduction technique that synergizes dynamic programming with a branch-and-bound exhaustive-search. In addition, we propose a novel infeasibility pruning technique $viz.$, *"Look Back"*, which benefits from NM's double pass design to further ease the constrained shortest paths finding in practice. Our NM is solving an NP-hard problem, so it is exponential in its general form. However, our computational complexity analysis shows that NM has a quadratically lower complexity upper-bound (halved exponent) than alternative methods. Moreover, when synergistically used with existing search space reduction techniques [8], [9], [10], our scalability evaluation results indicate how NM is faster by an order of magnitude than recent branch-and-bound exhaustive search methods, and hence, *scales* better.

Furthermore, NM is *flexible* due to its adaptable performance and applicability to different constrained shortest path scenarios with an arbitrary set of (SLO) constraints and an arbitrary cost function. For example, when we allocate traffic flow requests with a single path and multiple link constraints, NM can find some constrained shortest path variants in polynomial time. Thus, it can substitute diverse existing path finder algorithms such as the extended version of Dijkstra [20] and the iterative version of Bellman-Ford [12]. In its general form, NM can be also used to speed-up finding of all loop-free, k-constrained shortest or Pareto-optimal paths [9] from the source to the destination. Thus, NM is also applicable to diverse virtual network services including those with splittable and unsplitable flows. We demonstrate such flexibility with an extensive numerical simulation campaign, testing NM over diverse network topologies for both online VNE/NFV-SC with unsplittable flows and for TE with splittable flows. We found that the number of embedded VN requests and energy efficiency (and thereby the providers' revenue) can increase when the constrained shortest path management is used for tested VNE/NFV-SC solutions: either one-shot centralized ($i.e.$, with joined node and link embedding) [17], [18], [19] or two-stages distributed ($i.e.$, with separate node and link embedding) [21]. When using the constrained shortest path management within either linear programming [22] or greedy-based TE solutions [15], [16], our simulation results indicate gains of up to 50% in network utilization and lower energy consumption in some cases.

Finally, we implement an open-source Software-Defined Networking (SDN) based NM controller that is available at [23]. Our GENI [2] evaluation experiments with our implementation prototype confirm our analytical and empirical findings in real-world settings and show no constrained shortest path overhead (sought by NM) on the virtual network service management and data plane mechanisms at large scale.

Paper organization. In Section II, we formally state the constrained shortest path problem using optimization theory. Section III shows importance of the constrained shortest path management in VNE/NFV-SC and TE. In Section IV, we present details of our NM approach. The complexity improvements of our NM w.r.t. recent branch-and-bound exhaustive search algorithms are presented in Section V. Section VI describes our NM prototype implementation. Section VII describes our evaluation methodology, performance metrics and results. Section VIII concludes the paper.

II. The Constrained Shortest Path Problem

The constrained shortest path problem is the NP-hard problem of finding the shortest path subject to an arbitrary set of hop-to-hop and end-to-end constraints. In this section, we define this problem using optimization theory. In the subsequent section, we motivate the importance of its *flexible* and *scalable* management in diverse virtual network services.

A. Problem Overview

The problem of providing a (shortest) path with multiple (SLO) constraints is NP-hard [8], and its complete survey can be found in [11]. Herein, we mention a few representative solutions that help us present our novel contributions. Most heuristics group multiple metrics into a single function reducing the problem to a single constrained routing problem [24], and then solve the routing optimization separately, $e.g.$, using Lagrangian relaxation [25]. The exact pseudo-polynomial algorithm proposed by Jaffe $et\ al.$ [14] offers a distributed path finder solution limited to a two-path constraints problem. Wang $et\ al.$ [20] use an extended version of Dijkstra algorithm (EDijkstra), where all links with infeasible hop-to-hop constraints are excluded. EDijkstra runs in polynomial time but may omit any (path hop count) minimization, desirable in network virtualization to optimize the physical network utilization. To minimize the path hop count under a single path constraint ($e.g.$, delay) an iterative modification of Bellman-Ford (IBF) algorithm was proposed in [12]. Our approach is not limited to a single path constraint and can be adapted to subsume both EDijkstra and IBF as we discuss in Section V.

The authors in [10] propose an exact algorithm for the constrained shortest path problem, and apply several search space reduction techniques such as dominated paths (pruning by dominance and bound) and the look-ahead (pruning by infeasibility) notion for the exponential complexity exhaustive search, utilizing the k-shortest path algorithm. We also apply a similar technique to reduce the constrained path search space, though without any look-ahead, since it is computationally expensive. Instead, our design uses a more efficient "Look-Back" pruning technique (see Section IV-B). In [26], the authors propose an Exhaustive Breath-First Search (EBFS) based approach to solve the constrained path finder problem, focussing on delay. Another more recent work [9] also uses EBFS with a dominant path space reduction technique to find multi-criteria Pareto-optimal paths. Alternatively, an exhaustive Depth-First Search (EDFS) can be used as a branch-and-bound algorithm. For example, the authors in [8] proposed the "pulse" algorithm that uses EDFS with dominated paths and look-ahead search space reduction techniques. Both of EBFS and EDFS algorithms have exponential worst case time complexity. Our solution however quadratically reduces the worst case complexity of these algorithms (see Section V).

B. Constrained Shortest Path Problem

Let l be the number of hop-to-hop or *link constraints* for min/max network metrics, $e.g.$, bandwidth, and p be the number of end-to-end *path constraints* for additive/multiplicative network metrics, $e.g.$, delay or loss. Moreover, we denote with $l \oplus 1$ paths with multiple links and a single path constraints, and with and $l \oplus p$ paths with multiple links and multiple path constraints. Given the above notation, we define the constrained shortest path problem as follows:

Problem 1 (constrained shortest path)**.** *Given a physical network graph $G = (V, E)$, where V is the set of vertices and E the set of edges; let us denote with D the flow demand to be transferred and let u_{ij} denote a capacity of the directed edge e_{ij}; let f_{ij} be a binary variable $f_{ij} \in \{0, 1\}$ denoting a ratio of flow on the edge e_{ij}, and let c_{ij} denote a cost of transferring a unit of flow through such edge; finally, let $\bar{l}$ and $\bar{p}$ denote vectors of link (hop-to-hop) and path (end-to-end) constraints, excluding capacity constraints, where $\bar{l}$ corresponds to min/max edge e_{ij} weights $w^{\bar{l}}_{ij}$ (i.e., $\geq$ or $\leq$, respectively), and $\bar{p}$ corresponds to additive edge e_{ij} weights* [1] *$w^{\bar{p}}_{ij}$; the problem of finding constrained shortest path between source v_s and destination v_t vertices can be formulated as follows:*

$$\textit{minimize} \sum_{e_{ij} \in E} c_{ij} D f_{ij} \tag{1}$$

subject to

Flow Conservation Constraints:

$$\sum_{v_j \in V} f_{ij} - \sum_{v_k \in V} f_{ki} = \begin{cases} 1, & i = s \\ 0, & i \neq s \text{ or } t \\ -1, & i = t \end{cases}, \forall v_i \in V \tag{2}$$

Capacity Constraints:

$$D f_{ij} \leq u_{ij}, \forall e_{ij} \in E \tag{3}$$

Other Link Constraints:

$$w^{l}_{ij} f_{ij} \leq l, \forall e_{ij} \in E, l \in \bar{l} \tag{4}$$

Path Constraints:

$$\sum_{e_{ij} \in E} w^{p}_{ij} f_{ij} \leq p, \forall p \in \bar{p} \tag{5}$$

Existential Constraints:

$$f_{ij} \in \{0, 1\}, \forall e_{ij} \in E \tag{6}$$

Finding a *shortest path* (without constraints) has a polynomial time complexity: consider Equations 3, 4 and 5: in absence of any link or path constraints, the constraint matrix of the above optimization problem is unimodular [27]. This condition allows us to solve the optimization problem using (polynomial) linear programming. Such time complexity bound does not necessarily hold in presence of at least a single link or path constraint ($\bar{l} \neq 0$ or $\bar{p} \neq 0$). In that case, we have to solve the above optimization problem using (NP-hard) integer programming [27]. Note that we can always avoid specifying capacity constraints in Equation 3 by simply setting all $f_{ij} = 0$, which corresponding physical edge e_{ij} capacity u_{ij} is not sufficient to allocate flow demand D. The same applies for other link constraints (see Equation 3).

In the next section we show how finding a flexible and scalable solution to Problem 1 benefits a wide range of path finder subproblems to manage virtual network services.

[1] Note that multiplicative constraints (e.g., packet loss) can be converted to additive by composing them with a logarithmic function to avoid nonlinearity.

III. Constrained Shortest Path for Virtual Network Service Management

Using optimization theory, in this section we motivate the need for a flexible and scalable constrained shortest path to manage virtual network services such as Traffic Engineering (TE), Virtual Network Embedding (VNE) and NFV Service Chaining (NFV-SC). We also show how such a constrained shortest path scheme does not introduce any additional interoperability issues for aforementioned virtual network services with respect to traditional shortest path schemes.

A. Finding Virtual Paths in Resource Constrained Scenarios

We begin by considering a variant of the constrained shortest path that operates on a resource constrained scenario, $e.g.$, a natural or man made disaster scenario where connectivity is scarce. In those cases, one aim is to minimize the overall physical resource consumption of virtual paths. To this end, we define the *resource optimal constrained path* by modifying the objective of Problem 1 as follows:

Problem 2 (resource optimal constrained path)**.** *The resource optimal constrained path is a path that satisfies an arbitrary set of link/path constraints using minimal amount of physical bandwidth:*

$$\textit{minimize} \sum_{e_{ij} \in E} D f_{ij} \tag{7}$$

where f_{ij}, e_{ij}, D and E are as defined in Problem 1.

Note that by defining an equal weight c to all edges we seek the minimum hop path that satisfies an arbitrary set of link/path constraints ($e.g.$, imposed by SLO).

B. Traffic Engineering

TE techniques today can be roughly divided into two groups: oblivious $i.e.$, no a-priori knowledge of the SLO demands [28], and demands-aware, when such knowledge is available [15], [16]. Moreover, the later has a superior performance ($e.g.$, can better utilize substrate network resources) than the former [28] at the expense of having a centralized forwarding (or routing) control [15], [16], [29].

We broadly classify demands-aware traffic engineering solutions (see $e.g.$ [15], [16]) with the following network utility maximization problem:

$$\textit{maximize}[\min_{f_i \in F} fairness_i(f_i)] \tag{8}$$

where F denotes a set of all demands (or commodities); in [16], for example, such commodities are $\{src, dst, SLO\}$ tuples; $f_i \in [0, 1]$ is continuous variable that denotes the ratio of flow for commodity i with bandwidth demand D_i; and $fairness_i(f)$ is a linear piecewise-defined function whose definition is based on path service's demands SLO constraints. For a complete problem formulation we refer to [22], [30].

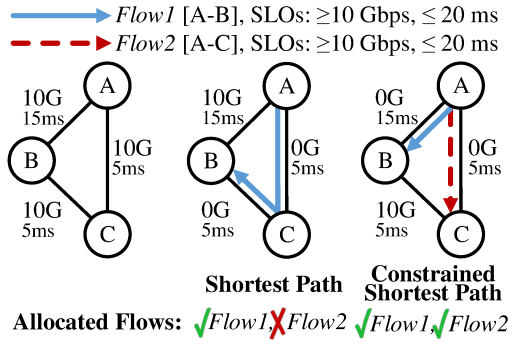


Fig. 1: Applying constrained shortest paths to allocate traffic flows improves network utilization: the widely applied shortest paths, *e.g.*, to allocate traffic flows, in this case $Flow1$, can hinder allocation of other flows; allocating $Flow1$ on a constrained shortest path as in Problem 2 instead permits the allocation of $Flow2$ as well.

Constrained shortest path relevance. There are two standard ways of formulating the TE optimization problem shown in Equation 8 — the arc-based [30] and path-based [22] formulations. In practice, due to hardware granularity limitations, the arc-based linear programming solution can be infeasible to implement, or require use of NP-hard integer programming which can be intractable even for moderate size networks. The shortest path algorithms such as Dijkstra (*e.g.*, within k-shortest path algorithms [31]) are currently used to find a set of paths as an input for the path-based linear programming formulations or for their simplified greedy solutions [15], [16]. The hope is also to map the highest priority flows to the minimum latency paths first. However, if we are aware of the services' SLO demands (*e.g.*, bandwidth, latency, loss rate, etc.), we can use them as constraints to optimize physical network utilization (and hence the flow fairness) as described in Problem 2.

Motivating example. Consider Figure 1: two flows $Flow1$ and $Flow2$ with bandwidth and latency constraints are to be allocated on a physical network of Figure 1-left. Attempting an allocation of $Flow1$ by merely considering the shortest path algorithm to minimize the latency (Figure 1-center) — as current TE solutions do, — the hosting physical path would have to use up to two physical links ($A \rightarrow C \rightarrow B$), thus preventing the allocation of the subsequent requested flow $Flow2$. If instead a constrained shortest path is used to allocate $Flow1$ and $Flow2$ as shown in Figure 1-right, then path $A \rightarrow B$ could be found, leaving capacity for the allocation of $Flow2$ (resulting in its higher fairness) as well.

On the contrary, when shortest path algorithms are used to find min-hop paths, such algorithms can lead to an infeasible solution even when a feasible one exists, leading to TE performance degradation. For example, consider the mapping of $Flow1$ with a latency constraint of 10 ms (tighter than 20 in Figure 1). The min hop (shortest) path $A \rightarrow B$ violates the 10 ms latency constraint, preventing the allocation of $Flow1$, whereas the constrained shortest path algorithms would find the feasible and optimal $A \rightarrow C \rightarrow B$ path even though more hops are needed to satisfy such tighter SLO demand. Note, however, that the above problem solution of allocating sequential flow requests can be suboptimal w.r.t. the problem solution of allocating such flows when they are known in advance. We dissect such network utilization gains when constrained shortest paths are applied for TE in Section VII.

C. Embedding Virtual Networks and Service Chains

Embedding a virtual network (service chain) requires a constrained virtual service request to be mapped on top of a physical network hosted by a single infrastructure provider, or by a federation of providers. To solve this (NP-hard [32]) graph matching problem, earlier work proposed centralized (see *e.g.* [17], [18], [19], [33], [34], [35], [36]) and distributed [21], [37], [38] algorithms. Such solutions either separate the node embedding from the link embedding phase [21], [33], [37], [38], or simultaneously apply the two phases [17], [18], [19], [36]. When link embedding is considered separately, paths can be found dynamically or precomputed for each virtual link using Dijkstra or k-shortest path algorithms [21], [33], [38]. When instead node and link embedding are considered jointly, recent work has shown how embedding problems can be formulated as known multi-commodity flow problems [17], [18], [19]. On the one hand, all these problems show how there is a need to minimize providers' management costs when allocating the virtual network or the service chain [18] as shown in the following objective:

$$\textit{minimize} \sum_{e_{st} \in E_V} \sum_{e_{ij} \in E_S} c_{ij} D_{st} f_{ij}^{st} + \sum_{v_s \in V_V} \sum_{v_i \in V_S} c_i D_s x_i^s \quad (9)$$

where V_S and E_S (V_V and E_V) denote sets of physical (virtual) vertices and edges, respectively; c_{ij} and c_i denotes unitary bandwidth and CPU cost on physical edge e_{ij}, respectively; $f_{ij}^{st} \in [0,1]$ is continuous variable that models the ratio of flow from v_s to v_t virtual vertices transferred through the physical edge e_{ij} when a flow with demand D_{st} is splittable, or, in its binary form, $f_{ij}^{st} \in \{0,1\}$ for unsplittable flows. Finally, $x_i^s \in \{0,1\}$ is the binary variable that denotes whether the virtual vertex v_s with computation demand D_s is assigned to the physical vertex v_i or not. For a complete problem formulation we refer readers to [17], [18], [19]. On the other hand, when virtual network (or service chain) requests are unknown in advance, the cost minimization strategy presented in Equation 9 can cause physical network partitioning, that in turn can lead to lower physical network utilization [17], [19]. To maximize the long term provider's revenue *i.e.* to allocate more virtual network requests, often a load balancing objective is sought [17], [19]:

$$\textit{minimize} \sum_{e_{st} \in E_V} \sum_{e_{ij} \in E_S} \frac{c_{ij} D_{st}}{u_{ij}} f_{ij}^{st} + \sum_{v_s \in V_V} \sum_{v_i \in V_S} \frac{c_i D_s}{u_i} x_i^s \quad (10)$$

where u_{ij} and u_i denote available capacities of physical edge e_{ij} and vertex v_i, respectively.

Constrained shortest path relevance. To cope with integer programming intractabilities for solving path-based multi-commodity flow problems, the well-known *column generation approach* can be applied as in [17], [18]. The column generation approach iteratively adds only those paths (*i.e.*, flow variables or columns) to the problem formulation that improve objective. In [17], [18] the Dijkstra shortest path algorithm is used to find the best shadow price paths (columns) for each virtual link to include them in the formulation. Although such approach best improve the objective value of *e.g.*, Equation 10, it can be suboptimal when virtual links have additional constraints such as latency or loss rate. In this scenario, a path between any two physical nodes (found

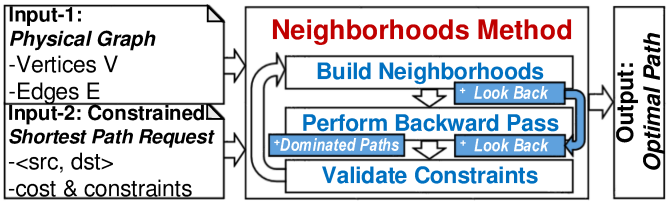


Fig. 2: Neighborhood Method Workflow in $l \oplus p$ case: The k^{th}-hop Neighborhood Build (forward pass) first (using label-correcting), and the Backward Pass later find all simple paths of k length (with any instance of branch-and-bound exhaustive search); in the third step, we check candidates feasibility; we then repeat recursively the backward pass with the $k+1$ neighborhoods, until the *optimal* (constrained shortest) path is found or all candidates have been eliminated. To improve the time to solution, we couple NM with a dominated path and a look-back search space reduction technique.

by solving Problem 1) can be a part of the optimal solution, even if it has a worse objective value than the shortest path, but satisfies all virtual link constraints. Generally, for any two stage or one-shot VNE algorithm, constrained shortest paths can be used to best improve the objective value while satisfying an arbitrary number of virtual link constraints. In Section VII, we confirm such intuition empirically, by studying the allocation ratio improvements (a proxy for provider's revenue) when our method is used to find paths for embedding services.

IV. Neighborhoods Method

To solve the NP-hard constrained shortest path problem with an arbitrary set of ($e.g.$, SLO) constraints in practical time, we propose the novel Neighborhoods Method (NM).

Why the name "Neighborhoods Method"? Most path seeking algorithms require at least two inputs for each node: (i) knowledge of neighbors, and (ii) awareness of all adjacent link costs, often dictated by policies, or SLO constraints. Such constraints are then used by the path seeking algorithm to compute the lowest-cost paths. The Dijkstra algorithm $e.g.$, recursively finds the shortest path traversing the source neighbors and the neighbors of their neighbors. This recursive notion leads to our definition of "neighborhoods" in NM, $i.e$, a set of nodes that can be reached from the source node with the same number of hops, where each "neighborhood" (being a set) contains unique elements. Based on Bellman's "Principle of Optimality" [39], such node repetitive structures can be used for label-correcting dynamic programming that we apply to reduce a number of exhaustive search path candidates.

A. The NM General Case ($l \oplus p$ case)

As the exact constrained shortest path algorithm, the worst case time complexity of NM when accepting an arbitrary set of hop-to-hop and end-to-end constraints ($l \oplus p$) is exponential. Our NM complexity analysis shows that the time complexity exponent is, however, halved with respect to common branch-and-bound path finders methods based on exhaustive search [8], [9], [10] (see Section V). Note also that in addition to the constrained shortest path, NM can simplify the process of finding all simple, k-constrained shortest or Pareto-optimal paths [9] from the source to the destination.

The general workflow of our NM algorithm is shown in Figure 2: NM is executed in three phases: (i) a *forward pass* or neighborhoods building (by using label-correcting),

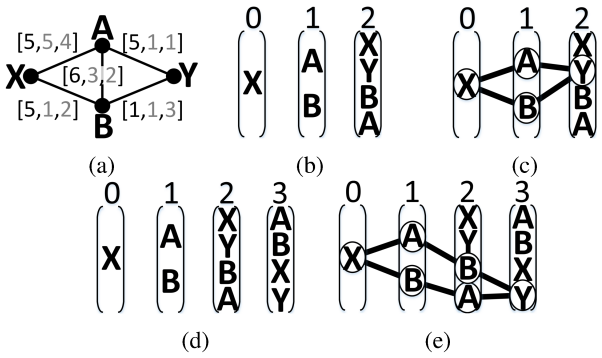


Fig. 3: Running example of NM in $l \oplus p$ case: (a) An example network configuration with [bandwidth, delay, cost] link metrics; (b) NM forward pass phase estimates the min hop count distance to Y during the first iteration, and (c) backward pass identifies all the shortest path candidates, which in this case do not contain the *optimal* path; (d) the forward pass adds more neighborhoods to find a path with longer distance to Y. (e) The Backward pass identifies all paths of a given new length; in this case the path contains the *optimal* $X \rightarrow B \rightarrow A \rightarrow Y$ solution.

(ii) a *backward pass* (with any instance of branch-and-bound exhaustive search), and a final (iii) *constraints validation* phase. During the forward pass, NM builds the neighborhoods to estimate the path length. The backward pass is used to find end-to-end paths with a given length (hop count). The final constraints validation phase is used to keep the best path candidate and decide whether or not the path search should be extended to longer path candidates involving more neighbors.

Example 1. *Consider a network consisting of 4 nodes X, Y, A and B, as shown in Figure 3a. On the link (X, A), we denote with the first value 5 the link constraint (in this case bandwidth), and with the second 5 we refer to the first path constraint (in this case end-to-end delay); with the third value 4 we refer to the second path constraint (in this case an arbitrary cost). In its general case, NM finds a path from X to Y satisfying the three constraints $bw \geq 5$, $delay \leq 5$ and $cost \leq 5$ as follows: in the first phase, NM builds all neighborhoods starting from the source node X until the destination node Y is reached (Figure 3b). Upon reaching Y, NM begins the backward pass to find the full set of min hop paths (Figure 3c). If the set contains the optimal path (in this case for Problem 2), NM terminates with a solution. If no such path is found, NM builds an additional neighborhood as shown in Figure 3d and performs another backward pass to check for optimality among paths that are one hop longer than at the previous iteration (Figure 3e). NM iterates until either the solution is found, or the maximum path length is violated. In this example, NM returns the constrained shortest path $X \rightarrow B \rightarrow A \rightarrow Y$.*

Forward pass for $l \oplus p$. Given an arbitrary graph $G(V, E)$, where $|V|$ and $|E|$ represent the number of vertices and edges, respectively, in this phase NM successively builds neighborhoods $< NH >$ from the source X to the destination Y. Algorithm 1 describes the forward pass of NM. To build a new neighborhood NH, we add therein neighbors (adjacent vertices) of each vertex u in the current neighborhood cNH (line 6). For example, the first NH includes nearest neighbors of the source X (which is in the zero NH), and the second NH contains nearest neighbors of vertices in the first NH, and so on. The first phase ends as soon as the destination Y

appears in cNH (line 4), or $< NH >$ size is more or equal to $|V|$ (line 13).

Algorithm 1: Build Neighborhoods ($l \oplus p$ case)

Input: X:= src, Y:= dest
Output: The list of neighborhoods $< NH >$ from X to Y

```
1  begin
2      cNH ⟵ X
3      < NH > ⟵ < NH > ∪ cNH
4      while Y ∉ cNH do
5          NH ⟵ ∅
6          foreach Vertex u ∈ cNH do
7              NH ⟵ NH ∪ adjacent(u)
8          end
9          if < NH > .size < |V| then
10             < NH > ⟵ < NH > ∪ NH
11             cNH ← NH
12         else
13             Return Y is unreachable.
14         end
15     end
16 end
```

Backward pass for $l \oplus p$. The best exhaustive search strategy may depend on the network topology, constraint and cost functions, and we leave it as a policy for NM. In this paper, we use EBFS for the backward pass of NM detailed in Algorithm 2. Note however, that this phase can use any exhaustive search ($e.g.$, EBFS [9], EDFS [8] or exhaustive k-shortest path [10]) with the only difference being that *we do not process all neighbors (adjacent vertices) of each vertex u but only those which are within previous NH* (line 10). The first step is to find the intersection between neighbors of the destination Y with its previous NH (line 5). This intersection is not an empty set, it contains at least one vertex v. For all obtained vertices we again build the intersection of their neighbors with the penultimate NH (line 16). The second phase ends as soon as we hit the zero NH (line 6), and as a result we obtain the collection of all paths with a length of $< NH >$ size between the source X and the destination Y.

Algorithm 2: Perform Backward Pass ($l \oplus p$ case)

Input: The list of neighborhoods $< NH >$ from X to Y
Output: All paths $< path >$ from X to Y of $< NH > .size$ length

```
1  begin
2      path ⟵ Y
3      < path > ⟵ < path > ∪ path
4      k ⟵ 1
5      NH ⟵ < NH > [size − k]
       /* EBFS:                                          */
6      while NH ≠ < NH > [0] do
7          < tempPath > ⟵ ∅
8          foreach path ∈ < path > do
9              Vertex u ⟵ path[1]
10             foreach Neighbor v ∈ adjacent(u) ∩ NH do
11                 < tempPath > ⟵ v ∪ path
12             end
13         end
14         < path > ⟵ < tempPath >
15         k ⟵ k + 1
16         NH ⟵ NH[size − k]
17     end
18 end
```

Constraints validation for $l \oplus p$. In this last phase, we check for candidates optimality, $i.e.$, we check whether or not a candidate path satisfies all $l \oplus p$ constraints, and keep the best candidate. At each consequent iteration, we first build an additional $(N+1)$ neighborhood, repeat the backward pass and subsequently obtain all paths of length $N+1$, and then we check their feasibility and update the best known path candidate, if needed. Similarly to IBF [12], we keep iterating while the candidate path length is less than the number of vertices $|V|$ and then return the optimal solution.

We close this subsection with three important remarks: **(1) NM can be used to find k-constrained shortest paths**: the backward path at each iteration returns all possible path candidates of the same hop count. To find k-constrained shortest paths we need to keep not a single best (shortest) path candidate, but a set of k best path candidates at each iteration and update this set if needed. Clearly, as in the worst case NM traverses all possible path candidates, its upper bound complexity does not change (see Section V). **(2) NM can be applied to both directed and undirected graphs making it an interesting solution even for NFV chain instantiations**. When traversing directed graphs, NM simply uses vertices' outgoing neighbors during the forward pass and incoming neighbors during the backward pass. **(3) A distinctive feature of our NM is the construction of the intersection of two neighborhoods**. This leads to a quadratic reduction of path candidates for exhaustive search algorithms (see Section V).

B. NM Search Space Optimizations

In this subsection we show how NM can be coupled with existing search space reduction techniques, $i.e.$, dominant paths or look ahead [8], [9], [10], to speed up its time to solution. By leveraging the NM's double pass, we also propose a variant of the look ahead technique, $viz.$, "Look Back" without complexity overhead. We first describe the *dominated paths* method. Observe that the dominant paths technique is not applicable in case we wish to use NM as k-constrained shortest paths, as it removes suboptimal candidates which can be among k paths.

Dominated paths (pruning by dominance or bound). The basic idea behind dominated paths states that an algorithm can avoid evaluating candidate paths when their (multidimensional) distance is longer than other candidates distance, since they cannot be a part of the shortest solution [9], [10]. Consider for example Figure 3a: note how path $X \rightarrow A \rightarrow Y$ with distance vector $\bar{d} = [6,5]^T$ is dominated by $X \rightarrow B \rightarrow Y$ path with distance vector $\bar{d} = [2,5]^T$ and hence it can be excluded to avoid unnecessary time-consuming processing. Path dominance is formally defined as follows:

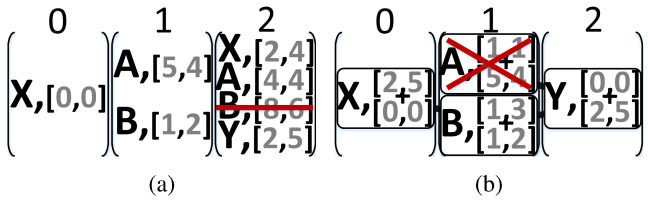
Definition 1 (dominant path)**.** *We say that path P_1 dominates path P_2 if and only if:*

$$\exists\, i \in 1,\ldots,p,c : d_i(P_1) < d_i(P_2) \wedge d_j(P_1) \leq d_j(P_2), \quad \forall j \neq i \in 1,\ldots,p,c$$

where d_i is a distance corresponding to p_i path constraint or to c path cost.[2]

In its general form, we can reduce NM's search space by removing the dominated paths. This is accomplished by keeping track of only non-dominated paths during its backward pass (Algorithm 2, line 11). In particular, a path is added to a vertex v if it is not dominated by any other path to v, or if it is not dominated by other paths from the source to the destination.

[2]Note how, if only the cost distance $d_c(P)$ is used in Equation 1, pruning by path dominance is the well-known pruning by bound technique [8].



(a) (b)

Fig. 4: NM running in its in $l \oplus p$ general case reduce its search space with a *Look Back*. (a) Forward pass: excludes vertices violating at least a path constraint; (b) Backward pass: NM looks back and removes a path candidate which sum of its current $delay$ and the best (estimated during the forward pass) residual $delay$ violates the $delay$ constraint ($i.e.$, $1 + 5 > 5$).

Look ahead (pruning by infeasibility). We can further reduce NM's search space by omitting path candidates with endpoint v, if the sum of their current path distances and the residual best distances from v to the destination violates any of the path constraints. This technique is known as "look ahead" [10] and requires a run of Dijkstra or Bellman-Ford algorithm p times to pre-compute best distances (corresponding to path constraints p) from all vertices to the destination.

Looking backwards (pruning by infeasibility). To avoid running $e.g.$, Dijkstra algorithm $p + 1$ times, in this paper we propose a more efficient look back technique that reduces the search space. The best distance could be estimated now from the intermediate vertex v to the source (instead of the destination) for each neighborhood containing v during the NM forward pass. We then use such information during the backward pass to exclude paths from v at neighborhood N that violate corresponding path constraints.

Note that when our looking backward technique is used, we can learn the best distance for each node in each neighborhood. Moreover, as physical nodes may appear in more than one neighborhood, we have more chances to prune path candidates than with the look ahead search space reduction technique. In addition, as we coupled the look back space reduction with the NM forward pass, we do not introduce additional overhead with respect to the look ahead reduction ($i.e.$, running Dijkstra $p + 1$ times).

Example 2. *In this example we revisit Example 1, to describe how NM in its the general case coupled with a Look Back search may reduce the search space when finding a path from X to Y. We assume that two path constraints $delay \leq 5$ and $cost \leq 5$ need to be satisfied. During the forward pass (Figure 4a) NM estimates the path length to Y, keeping track of all constraint-satisfying distances for each vertex and at each neighborhood. Vertices whose estimated distance violate at least one path constraint are instead removed. During the backward pass, NM removes $A \rightarrow Y$ path candidate whose sum of the current $delay$ (which equals to 1) and the best estimated $delay$ from the source to A (which equals to 5) violates $delay$ constraint $1 + 5 > 5$ (Figure 4b).*

C. NM for l links and a single path constraint ($l/l \oplus 1$ cases)

The worst case running time of NM becomes polynomial (by avoiding an exhaustive search) if either a general constrained shortest path (see Problem 1) with only l link constraints or the resource optimal constrained path (see Problem 2) with l links and a single path constraint ($l \oplus 1$) are sought (see Section V). This is because unnecessary iterations and phases (including the exhaustive search) are avoided. For the complete algorithm description in those cases, we refer reader to our earlier work on NM [40].

D. NM Optimal Solution

We conclude this section stating an important result, whose proof (by contradiction) is reported in our technical report [41].

Theorem 1. (Theorem of the Optimal Solution) *The Neighborhood Method always finds the optimal path if it exists.*

Proof. see technical report [41]. ■

V. Asymptotic Complexity Analysis

In this section, we provide a complexity analysis comparison among our NM and related algorithms such as: IBF [12], EDijkstra [20], EDFS [8] and EBFS [9], [26] — common branch-and-bound exhaustive search approaches. Table I summarizes the main results of this comparison, where $|V|$ is the total number of vertices, and $|E|$ is the total number of edges.

Theoretical results summary. The major benefits of NM arise when we are seeking the NP-hard [8], [9] constrained shortest paths in $l \oplus 1$ and $l \oplus p$ cases. We show how, under l constraints, EDijkstra finds constrained (shortest) paths faster than IBF and NM. We also show how NM is an alternative for IBF to find the resource optimal constrained path when accepting $l \oplus 1$ constraints (and thus *flexible*), and how EDijkstra loses any path optimality guarantees in that case. Finally, when accepting $l \oplus 1$ and $l \oplus p$ constraints, we show how time and space EBFS and EDFS complexities are quadratically higher with respect to our NM.

A. Complexity Analysis for the l and $l \oplus 1$ Cases

For lack of space and due to the fact that most of the $l/l \oplus 1$ complexity results are based on a simple extension of the original Dijkstra, Bellman-Ford, DFS and BFS algorithms, we only show complexity analysis for the general $l \oplus p$ case. The in-depth complexity analysis for $l/l \oplus 1$ cases however can be found in our technical report [41]. Note how the constrained shortest path in the $l \oplus 1$ case is NP-hard [8] and can be found only by the general $l \oplus p$ case algorithms.

B. Complexity Analysis for the $l \oplus p$ Case

EDijkstra and IBF complexities ($l \oplus p$ case). For completeness, we mention that neither EDijkstra nor IBF are applicable to any variant of the constrained shortest path in the $l \oplus p$ case.

EDFS and EBFS complexities ($l \oplus p$ case). Both EDFS and EBFS can find any variant of the constrained shortest path or k such paths in exponential time by exhaustive search. For each potential path, these algorithms check the satisfaction of all p constraints by calculating $p + 1$ new path and new cost metrics in $O(2p)$. All l constraints violating edges could be pruned prior to running EDFS or EBFS. The total number of path candidates can be bound as $O(b^d)$ [42], where b is the number of neighbors, and d is the maximum loop-free path

TABLE I: Virtual path embedding algorithms complexity

Case:	EDijkstra [20]	IBF [12]	EDFS [8], EBFS [9], [26]	NM
l	Time: $O(\|V\|+\|E\|\cdot l)$ Space: $O(\|V\|+\|E\|)$ *resource optimal constrained path*	Time: $O(\|V\|+\|E\|\cdot l)$ Space: $O(\|V\|+\|E\|)$ *resource optimal constrained path*	Time: $O(\|V\|+\|E\|\cdot l)$ Space: $O(\|V\|+\|E\|)$ *resource optimal constrained path*	Time: $O(\|V\|+\|E\|\cdot l)$ Space: $O(\|V\|+\|E\|)$ *resource optimal constrained path*
$l/l\oplus 1$	Time: $O(\|V\|log\|V\|+\|E\|\cdot l)$ Space: $O(\|V\|+\|E\|)$ **constrained shortest path/** constrained path only	Time: $O(\|V\|\|E\|+\|E\|\cdot l)$ Space: $O(\|V\|\|E\|)$ **constrained shortest path/** *resource optimal constrained path*	Time: $O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\|V\|}+\|E\|\cdot l\big)$ Space: $O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\|V\|}\big)$ **constrained shortest path**	Time: $O(\|V\|\|E\|+\|E\|\cdot l)$ Space: $O(\|V\|\|E\|)$ **constrained shortest path/** *resource optimal constrained path*
$l\oplus p$	N/A	N/A	Time: $O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\|V\|}p+\|E\|\cdot l\big)$ Space:$O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\|V\|}p\big)$ **constrained shortest path**	Time: $O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\frac{\|V\|}{2}}p+\|E\|\cdot l\big)$ Space: $O\big(\big(\frac{\|E\|}{\|V\|}\big)^{\frac{\|V\|}{2}}p\big)$ **constrained shortest path**

hop count. The average number of neighbors per vertex b can be obtained from the hand-shaking lemma[3]:

$$b=\frac{\sum_{i=v}^{V}(neighbors\ of\ vertex\ v)}{|V|}=\frac{2|E|}{|V|}. \quad (11)$$

Using Equation 11 and based on the fact that the maximum loop-free path hop count equals to $|V-1|$, EDFS and EBFS time complexities $O^{l\oplus p}$ are:

$$O^{l\oplus p}=O(2p\cdot b^d+|E|l)=O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|}p+|E|\cdot l\Big) \quad (12)$$

The space complexity equals to $O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|}p\Big)$ due to the fact that we have to store p metrics for each path.

NM complexity ($l\oplus p$ case). Similar to EDFS and EBFS, NM can find any variant of the constrained shortest path or k such paths with $l\oplus p$ constraints in exponential time utilizing any instance of an exhaustive search. The neighborhoods contains non-unique nodes, and the total number of their nodes can be up to $|V|^2$. For each neighbor, NM checks if it already appears in the neighborhood $O(b)$. Taking into account the edge pruning phase, to ensure the constraints satisfaction and to reduce a search space, the time complexity of the neighborhoods building step $O_1^{l\oplus p}$ is:

$$O_1^{l\oplus p}=O(|V|^2b+|E|l)=O(|V||E|+|E|\cdot l) \quad (13)$$

During the backward pass, NM builds all possible paths from the destination node using any exhaustive search methods such as EDFS or EBFS. *However, there is a difference in that, we process only those vertex neighbors which appear in its previous neighborhood.* This allows us to significantly reduce the total number of the processing paths: instead of processing $1+b+b^2+...+b^d$ or $O(b^d)$ paths, due to double pass ($i.e.$, forward and backward passes) we process only: $1\cap b^d+b\cap b^{d-1}+b^2\cap b^{d-2}+...+b^{\frac{d}{2}}\cap b^{\frac{d}{2}}+...+b^{d-2}\cap b^2+b^{d-1}\cap b+b^d\cap 1\leq 1+b+b^2+...+b^{\frac{d}{2}}+...+b^2+b+1$ or $O(b^{\frac{d}{2}})$ paths. Hence, the time complexity of the backward pass step $O_2^{l\oplus p}$ is quadratically lower than for EDFS or EBFS (see Equation 12):

$$O_2^{l\oplus p}=O(2p\cdot b^{\frac{d}{2}})=O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}}p\Big) \quad (14)$$

The total time complexity of the NM is $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}}p+|V||E|+|E|\cdot l\Big)$ or just $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}}p+|E|\cdot l\Big)$. Similarly as for EDFS and EBFS, the total space complexity is $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}}p\Big)$.

[3]Without loss of generality, we can assume undirected network graphs as in a directed graph, b equals to the average outdegree: $b=\frac{|E|}{|V|}$.

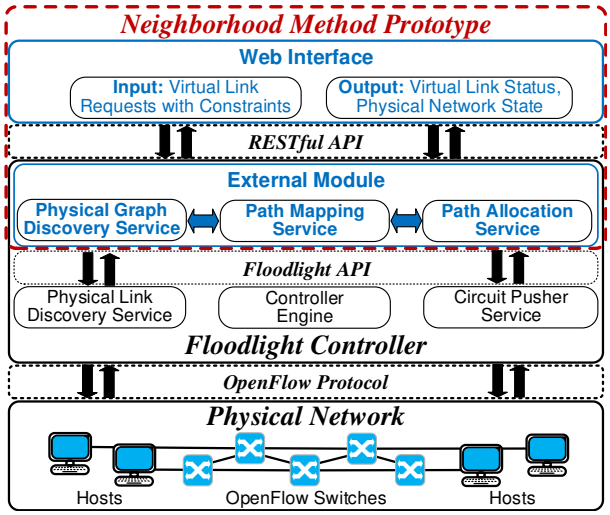


Fig. 5: System architecture of our NM prototype (which is a module of the Floodlight OpenFlow controller [43]) includes four main logical components: a *physical graph discovery* service, a *path mapping* service, a *path allocation* service and a user *web interface* that interacts with the controller. The prototype source code is publicly available under a GNU license at [23].

VI. NM Prototype Implementation

In this section, we establish the practicality of our approach for network virtualization with a prototype implementation over a Software Defined Networking infrastructure. Our source code is publicly available at [23]. In particular, our prototype implementation extends the Floodlight OpenFlow controller [43]. Our system architecture is shown in Figure 5. Our prototype includes four main logical components: a *physical graph discovery* service, a *path mapping* service, a *path allocation* service and a user *web interface* that interacts with the controller. In the rest of the section we describe with some more details each of the four components of our prototype.

Physical graph discovery. Upon bootstrapping a SDN setup, all configured OpenFlow switches connect to the controller allowing a dynamic *switch-to-switch* link discovery. After this phase, the NM module tracks all unknown incoming packets to detect hosts and their corresponding *host-to-switch* links. We specify the main physical link properties, such as capacity and cost ($e.g.$, delay) through an XML configuration file. The XML configuration file indirection allows our NM prototype to easily interact with foreign measurement services, for example for real-time awareness of its link properties. The information

collected by the *path discovery* service is then used in the *path mapping* and the path *allocation* steps.

Path mapping. To map constrained virtual link requests, the path mapping module of our prototype uses information from the physical path discovery service and runs one of the following routing algorithm policies: NM for l and $l \oplus 1$ cases (default policy), NM for $l \oplus p$ cases, EDijkstra, IBF or EBFS. In the current version of our prototype the routing policy is static and has to be set via our XML configuration file before starting the Floodlight controller.

Path allocation. In this last phase of the path embedding, the NM module sets all appropriate flow rules in all switches via the OpenFlow [44] protocol, along with the computed path obtained during the path mapping phase. Specifically, using OpenFlow protocol messages the module assigns active flows to corresponding queues, $i.e.$, applies an *ENQUEUE* action, for guaranteed bandwidth provisioning. In our XML configuration file, users may specify also the type of timeout for each flow $i.e.$, the flow duration time can start from either the previously matched packet or from the first flow instantiation. To estimate the available bandwidth on each physical link, the NM module uses both the capacity information derived from the XML configuration file, and its allocated bandwidth queried from the flow stored in the OpenFlow switch.

Web interface. To request virtual links and/or to monitor physical network states, we have implemented a web user interface, which uses a RESTful API to communicate with our NM prototype module. The user interface uses technologies such as HTML, PHP, AJAX, JavaScript, and CSS.

VII. Performance Evaluation

In this section, we evaluate NM's scalability and flexibility performance through simulations and our prototype implementation in the context of its applicability to several complementary virtual network services. To assess the flexibility of NM, we compare, with and without it: (i) the embedding performance of several online VNE and real-time NFV-SC mechanisms within the management plane; (ii) several traffic engineering solutions within the data plane. Our goal is to show how our NM can be used to improve the overall network utilization (allocation ratio or total flow throughput), optimality (load balancing or fairness of flows), as well as energy consumption within both planes; (iii) To assess NM scalability, we then compare it with related solutions under different network scales and service requests/topology scenarios when accepting multiple link and multiple path constraints; (iv) Finally, we confirm our main simulation results with our NM prototype running over the GENI testbed [2].

Evaluation settings. In our simulations, we used a machine with an *Intel Core i5* processor with dual core CPU of 2.7 GHz and 8GB RAM. We use the BRITE [45] topology generator to create our physical and virtual networks. Our results are consistent across physical networks that follow Waxman and Barabasi-Albert models [46], that are known to approximate well subsets of Internet topologies [47]. For lack of space we only show results relative to Waxman connectivities. In our NM prototype evaluation instead, we use a physical network obtained with the GENI testbed [2]. All our results show 95% confidence interval, and our randomness lays in both the service request ($i.e.$, in its type and constraints to accept) and in the physical network topology. In most of our physical topologies, the average node degree equals to 4, a known common value within Internet topologies [47].

Results summary. *Efficiency and Provider's revenue:* During the virtual network formation (management plane), we found that using NM within recent VNE or NFV-SC algorithms increases their allocation ratio while improving the network utilization by better load balancing (close to the optimal), which in turn decreases energy consumption. *Network Utilization:* Our results evaluating NM within the data plane instead show how utilizing a set of paths found with NM is beneficial for TE in terms of minimum path hop count, network utilization and in some cases even energy consumption. *Time to Solution (Convergence Time):* When we attempted to allocate flows (virtual links) with multiple link and multiple path constraints over large scale physical networks using NM with the proposed search space reduction techniques, we found a path computation speed-up of almost an order of magnitude w.r.t. common exhaustive search algorithms. Moreover, we also found almost 3 orders of magnitude running time improvement w.r.t. the same integer programming problem solved with CPLEX [50]. *Prototype Evaluation:* Finally, using our NM prototype over GENI, we were able to reproduce our main results. Moreover, our measurements show how the constrained shortest path computation (virtual link or flow mapping phase in NM) is up to an order of magnitude faster than the path allocation phase ($i.e.$, setting up appropriate flow rules within all switches along the found path) on small scale networks. This along with our scalability results confirm how at large scale the time needed for a path computation with NM will have the same order of magnitude as the time needed for the path allocation introducing no significant bottleneck for the end-to-end virtual link embedding.

A. Management Plane Evaluation

Simulation settings. To assess the impact of NM on the virtual network embedding, we include results obtained with simulated physical networks of 20 nodes (as in similar earlier studies [17]), following Waxman connectivity model, where each physical node and each physical link have uniformly distributed CPU and bandwidth from 0 to 100 units, respectively. Note that we use fairly small scale physical networks due to complexity of the integer programming formulation. We attempt to embed a pool of 40 VN (service chain) requests with 6 virtual nodes and random (linear) virtual topologies. Each virtual node and each virtual edge have uniformly distributed CPU and bandwidth demand from 1 to 10 units, respectively. When we tested NM within virtual network embedding algorithms, we vary the virtual node degree from 1 (the VN has linear topology) to 5 (VN is a fully connected topology). Moreover, we also assume that each virtual edge has a propagation delay stretch latency constraint uniformly distributed between 1 to 4. We defined propagation delay stretch the ratio between the propagation delay and the propagation delay encountered traversing the diameter of the physical network. When we tested NM within the real-time service chaining use case, we vary the latency constraint of virtual links (service-to-service communication) from ∞ ($i.e.$, SC is not real-time sensitive) to 1/4 ($i.e.$, SC is highly real-time sensitive) of the propagation delay stretch.

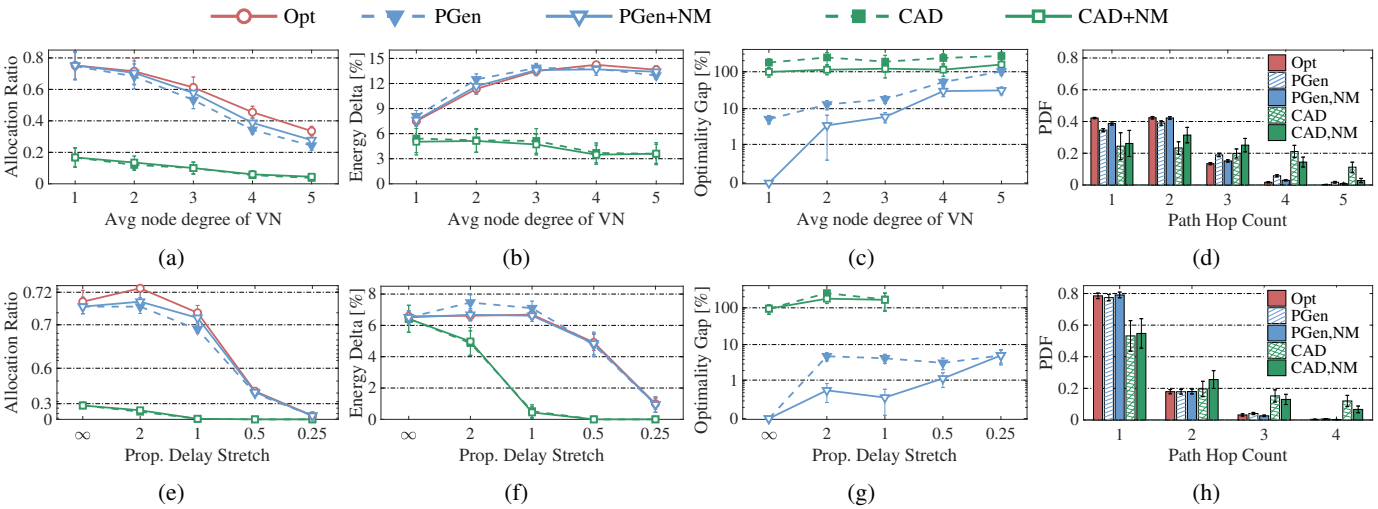


Fig. 6: Virtual network (VN) embedding and real-time NFV service chaining (SC) results obtained with physical networks of 20 nodes following Waxman connectivity model: by addressing the constrained shortest path problem with NM versus using commonly adopted shortest path algorithms ($e.g.$, Disjktra), the allocation ratio of VNE (a) and real-time NFV-SC (e) can be improved when the virtual node degree increase or when requests are not highly sensitive to delays. Optimality gap of VNE (c) and NFV-SC (g) can be also improved (resulting in our case into a better load balancing) which leads to a lower energy consumption (b) and (f) relative to the network idle state [49], respectively. NM's benefits are due to its ability of finding more path with a lower hop count that can satisfy latency demands and simultaneously improve objective value for full-mesh VNs (d) and moderate real-time sensitive SC (h) pools.

Evaluation metrics. To demonstrate the advantages of using NM within the virtual network embedding (VNE) and real-time service chaining (SC) mechanisms, we compared four representative VNE algorithms. To compare them, we replace their (original) Dijkstra-based shortest path with our NM. We have chosen to compare against [17], [19] as the optimal VNE scheme formulated as integer programming multi-commodity flow is the best to our knowledge solution for online VNE (yet intractable for large scale networks). We refer to this solution as Optimal and we denote it as Opt. Note however, that Opt solution can be still suboptimal with respect to the optimal solution of the offline VNE problem, where all requests are known in advance. Opt in fact attempts to minimize the ratio between the provider's costs of embedding a VN request and the available substrate resources provided for this request, with the aim of balancing the network load. We also compare Opt against its version where a path (or column) generation approach is used to make Opt more scalable [17]. We refer to this scheme as PathGen and, even in this case, substitute its original shortest path algorithm (used to find new paths within the multi-commodity flow) with our path management solution (see details in Section III-C). Note that we used the one-shot VNE approximation algorithm proposed in [36] as an initial solution for the column generation approach to avoid two stage VNE limitations when physical network is initially unbalanced [17]. Finally, we compare against a Consensus-based Auction mechanism (CAD) [5], [21], the first policy-based distributed VNE approximation algorithm with convergence and optimality guarantees. Note that a version of CAD can be also used to solve the NFV-SC problem [5]. The link embedding of CAD is a policy that runs a shortest path algorithm to either pre-compute the k-shortest paths or to find these paths dynamically. For fairness of comparison, we assume that the latter holds and as in the PathGen case, we substitute the currently used shortest path algorithm, Dijkstra, with our NM constrained shortest path finder solution.

In this simulation scenario we have tested the potential revenue loss by specifying the fraction of VN request accepted over the VN requested (allocation ratio), to what extent physical links were utilized (link utilization) and how many paths of the particular length in total were used per VN pool. Finally, we used the idle energy model proposed previously [49] to access the energy consumption of the network:

$$\text{Energy Consumption} = \sum_{e \in E} (M - E_0) U_e + E_0 \qquad (15)$$

where E is a set of physical edges, U_e is an edge e utilization, and M and E_0 are numerical values taken from [49] that correspond to the maximum and idle energy consumption of the switch interface, respectively. We use $M = 2$ and $E_0 = 1.7$ maximum and idle energy consumptions (measured in Watts) assuming gigabit channel communications. In our results, we show an energy consumption increase relative to the idle network state (in %).

NM improves VNE/NFV-SC allocation ratio and energy efficiency. Figures 6a and 6e show how including constrained shortest paths ($e.g.$, found with NM) during column generation of the PathGen approach can improve overall VNE and NFV-SC acceptance ratio. Particularly, the highest acceptance ($i.e.$, allocation) ratio gains arise in the case ofdense (e.g., full-mesh) VNs or under moderate real-time sensitivity of NFV-SCs. Moreover, we can see how in some cases, utilizing NM within PathGen leads to a lower energy consumption (see Figures 6b and 6f). This is due to an improved network utilization because of a better (closer to the optimal) load balancing (see Figures 6c and 6g).

These optimality gains in turn arise due to NM's ability to find more paths that can satisfy all virtual link constraints ($e.g.$, bandiwdth and latency) and simultaneously improve the objective value (see Figures 6d and 6h). These results confirm our expectations in Section III-C. At the same time, optimality improvements with NM demonstrate no significant benefits (in terms of allocation ratio or energy efficiency) for the CAD over standard shortest path management. This is due to the fact that in our settings, separate node and link embedding approach

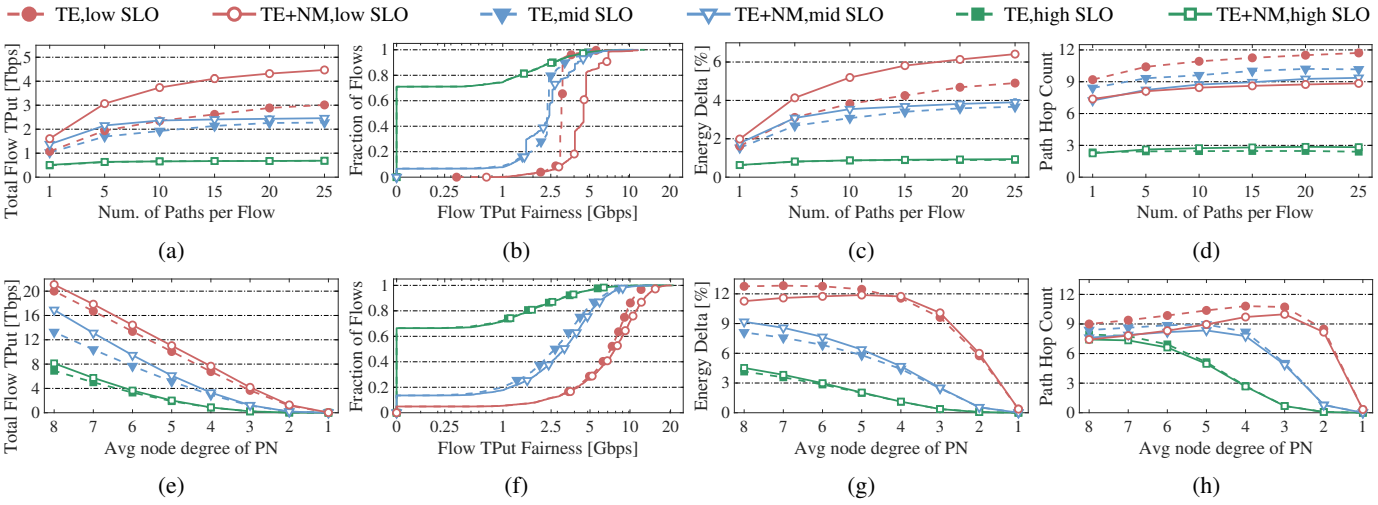


Fig. 7: Performance analyses of LP-based (*Top*) and greedy (*Bottom*) max-min fairness Traffic Engineering (TE) algorithms [15], [16] utilizing the constrained shortest path management with NM versus their original shortest path management with Dijkstra on Waxman topologies in terms of: (a) and (e) total gained flow throughput; (b) and (f) cumulative distribution of flow throughput for 25 paths per flow and for average node degree 4 (common for Internet [47]), respectively; (c) and (g) energy consumption increase relative to the network idle state [49]; and (d) and (h) number of average path hops per flow.

demonstrates the worst performance caused by significantly limited feasible space for the virtual link mapping. Such limitations are due to randomized capacities of physical nodes and edges ($i.e.$, due to initially unbalanced physical network) further exacerbated by randomized virtual link capacity and latency constraints.

B. Data Plane Evaluation

In the next set of results we analyze the benefits of using our solution within the data plane by evaluating NM performance within standard Traffic Engineering schemes [15], [16], under different physical network topologies and under different severity of service level objectives.

Simulation settings. To evaluate the impact of NM (used to compute constrained shortest paths) within Traffic Engineering [15], [16], we use a physical network topology of $10,000$ nodes, where each physical link has bandwidth uniformly distributed between 1 and 10 Gbps. We attempt to allocate flows for 1000 random source destination pairs by solving max-min fairness problem shown in Equation 8 with the fixed latency SLO demands. To evaluate the maximum possible gains, we assume infinite bandwidth demands of the flows and we omit any constraints imposed by hardware granularity due to rule count limits or flow quantization limitations [15], [16]. For clarity, we also assume that all flows have the same priority. Thus, the fairness of the flow is its total allocated throughput. We denote with low, medium and high delay SLO constraints, 4, 1.5, and 1 times of a propagation delay stretch defined in Section VII-A, respectively.

In the first simulation scenario, we use LP-based solution (that is costly to address in practice [15], [16]) with fixed average physical node degree equal to 4 (common for the Internet [47]), where we vary the maximum number of paths available for each flow allocation. In the second scenario, we use instead scalable greedy solution proposed in [16] with unrestricted number of paths per flow. To this end, once the best currently available path (or tunnel) gets fully saturated, we find the next best path dynamically. In this scenario, we vary average physical node degree from 8 to 1.

Evaluation metrics. To evaluate the performance of NM for data plane TE solutions, we compare NM with (extended) Dijkstra algorithm within the greedy-based TE ($i.e.$, in the second scenario), and their corresponding k-shortest path [31] and k-constrained shortest path (that uses general version of NM coupled with *Look Back* technique) algorithms within LP-based TE ($i.e.$, in the first scenario). We remark that NM's superior performance w.r.t. Dijkstra-based TE is expected due to its ability of minimizing provider's associated cost for flow allocation $e.g.$, minimizing provisioned physical bandwidth (see Problem 2) under an arbitrary set of ($e.g.$, SLO) constraints. This difference is expected to degrade in the first scenario (where LP-based formulation is used) with either number of maximum paths per flow or SLO severity increase, as in this case the k-shortest path set converge to the k-constrained shortest path set resulting in the equal LP formulation. However, in the second scenario, as we allocate bandwidth for flows on their most preferable tunnels (best paths) first, that superior performance is expected to be preserved and can vary with different node degree or SLO severity. For comparison we use the following four metrics: total gained throughput of all flows, cumulative distribution of flows' throughput which corresponds to their fairness, energy consumption relative to the network idle state (see Equation 15) and path hop count.

Path hop savings lead to network utilization and flow fairness gains. In Figure $7a$, we show how the correlation between the total gained throughput across all flows and the maximum number of paths available per each flow is a logarithmic function — increasing number of paths linearly brings a logarithmic growth to the total gained throughput. Figure $7e$ shows how the total gained throughput of all allocated flows changes when multiple physical links become available (the average physical node degree increases). This dependence is an affine function: the maximum possible total gained throughput increases linearly with the available physical links.

In both scenarios, we can see how the total flow throughput and the resulting flow fairness ($e.g.$, the particular flow throughput) are higher for NM than for Dijkstra-based TE

(see Figures 7a, 7b, 7e and 7f). These results demonstrate how minimizing the total physical bandwidth provisioned for a single flow with NM can significantly benefit ~~even~~ traffic engineering solutions. In particular, due to the path hop count optimization under SLO constraints within the data plane, NM gains up to 50% of total flow throughput under low SLO in the first (LP-based) scenario, and up to 20% of total flow throughput under mid SLO in the second (greedy-based) scenario w.r.t. Dijkstra-based TE (see Figures 7a and 7e). Such gains allow, in turn also to improve flow fairness (see Figures 7b and 7f). Note that such large throughput gains in the first scenario ($i.e.$, up to 50%) are partly due to the k-shortest path [31] and the general version of NM (that finds k-constrained shortest paths) algorithms difference, $i.e.$, the former has a higher probability of finding paths with more shared edges then the later. As expected, NM gains decrease with the node connectivity, as less physical path choices are available to map virtual links. Also, for LP-based scenario these gains decrease with increase of the maximum number of paths or SLO severity, as both shortest and constrained shortest path sets converge to each other resulting in equal LP formulations.

Average path length and energy consumption tradeoff. We further investigate the reasons why we observed such gains in total throughput of all flows w.r.t. Dijksra-based TE. In particular, observing Figures 7d and 7h we note that there are $\approx 2-3$ hops difference in the average path length between NM and Dijkstra-based TE when allocating low SLO flows. At the same time, for the medium SLO constraints this difference is reduced to circa one hop. Finally, when the SLO constraints are high, there is no significant physical path length difference. To understand why the average path length changes with the constraint severity, note how the longer is an end-to-end physical path, the lower is the probability that the entire path satisfies the SLO constraints. On the other hand, the higher the number of hops, the higher is the number of candidates paths, and so the higher is the probability of finding one which satisfies these constraints. This explains the trade-offs in average path length behavior observed in Figure 7h.

The hop count savings minimize the physical bandwidth provisioned for a single path, allowing the provider to accept more flows or allocate more bandwidth for a single flow. As a result, the overall link utilization increases leading to a higher energy consumption (see Figures 7c and 7g). We observe one exception when throughput gains are low and path hop count savings are high (as observed in the management plane scenario in Section VII-A). An example of such situation can be also observed in the second scenario for dense physical networks (with average node degree ≥ 5) when allocating flows with low SLO demands. In that case, we can see a small reduction (of $\approx 2\%$) in the energy consumption simultaneously with low throughput gains (of $\approx 5\%$).

C. Scalability Results

In the next set of results we test the scalability performance of NM when accepting multiple link and multiple path constraints ($l \oplus p$ case). We remark that in this case only exponential exact solutions exist for the constrained shortest path problem due to its NP-hardness [8].

Scalability simulation settings. To assess NM scalability, we simulate on-line requests for allocating constrained virtual

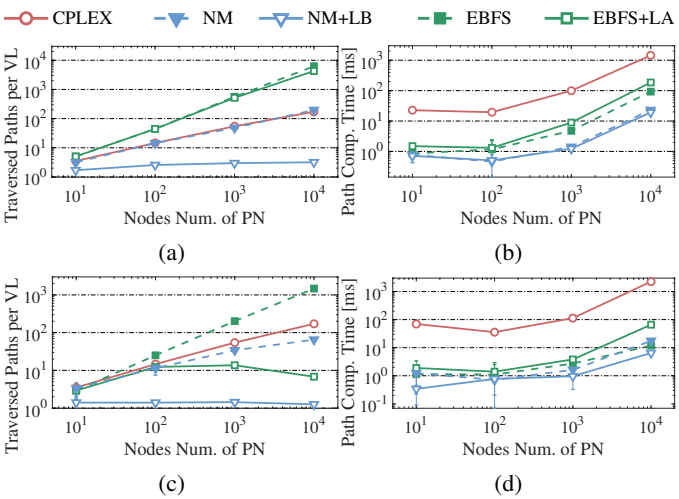


Fig. 8: Performance analyses of general NM versus EBFS with dominant paths, look-back for NM (NM+LB) and look-ahead for EBFS (EBFS+LA) search space reduction techniques, and versus IBM CPLEX solver for the constrained shortest path formulation in Problem 2 for low (top row) and medium (bottom row) SLO constraints in terms of: (a,c) number of traversed paths to find the *optimal* virtual path; (b,d) the virtual path computation time.

links (or traffic flows). In particular, we generate physical network topologies of 10, 100, $1K$ and $10K$ nodes, where each physical link has bandwidth uniformly distributed between 1 and 10 Gbps. In addition, we set each physical link with a cost uniformly distributed between 1 and 10. We attempt to find the constrained shortest path variant — the resource optimal constrained path for 10% of physical network nodes random source-destination pairs, where for each pair we allocate as many virtual links as possible with the fixed demands. We denote with low and medium bandwidth constraints, 1 and 4 Gbps, respectively; these values represent approximately 10% and 45% of the maximum physical link capacity. Similarly, we denote with low and medium (propagation) delay SLO constraints, 4 and 2.5 times of a propagation delay stretch defined in Section VII-A, respectively. In addition, we denote with low and medium cost constraints, 100 and 50 that represent 10 and 5 times of the maximum physical link cost.

Scalability evaluation metrics. To evaluate the NM scalability, we compare the general version of NM with the EBFS (common branch-and-bound exhaustive search) algorithm and with the CPLEX [50] performance (that uses 4 parallel threads) of solving the common arc-based constrained shortest path formulation. Both NM and EBFS are coupled with dominant paths search space reduction techniques. Moreover, we couple EBFS with a Look Ahead (EBFS+LA) search space reduction technique [10] and NM with a Look Back (NM+LB) search space reduction technique - a variant of Look Ahead without complexity overhead (see Section IV-B).

We compare NM with EBFS and CPLEX across two metrics: the number of traversed paths required to find the constrained shortest path and the average path computation time. Note that in case of CPLEX, the number of traversed paths corresponds to the total number of iterations.

Dominant paths prevent intractabilities. Figures 8a and 8c show that dominant paths technique reduces the number of traversed paths per virtual link to a linear function of a physical network size for both EBFS and NM. Moreover, due to its "double pass" technique, NM traverses up to two orders of magnitude paths less then EBFS. However, we can

see how EBFS works slightly faster than NM for large scale physical networks with medium link and path constraints (see Figure $8d$). That can be explained with the more expensive forward pass of NM for larger physical networks ($\geq$ 10K nodes). The NM's unnecessary iterations can be however reduced by the proposed Look Back technique.

NM scalability with backward pass and look back. Our experiments show that when NM backward phase is coupled with a Look Back technique, the number of traversed paths by NM further reduces. This reduction is almost independent from the size of the physical network (see Figures $8a$ and $8c$). Using the Look Back search space optimization does not introduce any significant path computation overhead, while the same cannot be said for the Look Ahead search space reduction technique (see Figures $8b$ and $8d$).

Even though CPLEX uses 4 parallel threads (instead of a single thread for NM and EBFS) and traverses moderate number of path (similar to NM without Look Back technique), it shows the worst performance in all cases. That is due to the fact, that finding constrained shortest paths with the commonly utilized arc-based integer programming formulation is NP-hard and no existing techniques can reduce that complexity to pseudo-polynomial. On the contrary, NM and EBFS complexities can be reduced to pseudo-polynomial by applying the dominant paths search space reduction technique [10]. Proposed novel *double pass* and *Look Back* search space reduction techniques further reduce the practical complexity of finding constrained shortest paths. Thus, NM is almost an order of magnitude faster in comparison with EBFS and is almost 3 orders of magnitude faster than CPLEX for large-scale physical networks, and hence *scales* better. We remark that such *scalability* improvements over existing constrained shortest path algorithms are essential for the virtual network service management at large scale. For example, the column generation approach can generate tens of thousands paths per a single VN request at large scale. Thus, it will take 100 ms x $10K \approx 17$ minutes when EBFS is used. On the contrary, we need only 10 ms x $10K \approx 100$ seconds with NM which significantly reduces VN request blocking probability. Note that additional scalability results comparison of our NM with respect to the EDijkstra shortest path scheme can be found in our prior work [40].

D. Prototype Evaluation

In this final set of results, we use our NM prototype to estimate the impact of the on-demand constrained shortest path computation on the end-to-end virtual link embedding performance. We also confirm our main simulation results.

Experiment settings. Our setup for the performance experiments includes 15 virtual machines (VMs) from the GENI testbed [2]: Ten of these VMs are OpenFlow Virtual Switches (OVS) [51], and others are hosts. Each *host-to-switch* physical link has 10 Mbps bandwidth and a 0 arbitrary cost, and each *switch-to-switch* physical link has both bandwidth (measured in Mbps) and an arbitrary cost uniformly distributed between 1 and 10. Note, that our arbitrary cost is an additive metric and therefore can represent any path metric, $e.g.$, delay, losses, jitter, etc. We request virtual links with low SLO constraints, $i.e.$, $\geq$ 1 Mbps bandwidth and $\leq$ 50 arbitrary cost (5 times greater than the maximum physical link cost), between 5

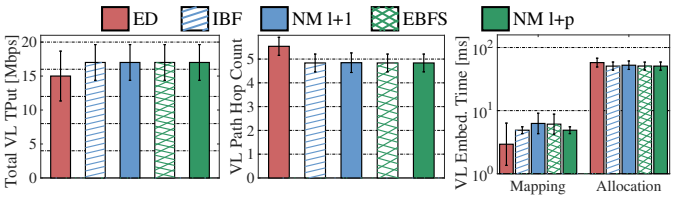


Fig. 9: Performance analysis of the shortest path algorithm such as the extended version of Dijkstra (ED) versus constrained shortest path algorithms such as NM (in both $l \oplus 1$ and $l \oplus p$ cases), EBFS and IBF on a reserved in GENI small SDN testbed in terms of: (a) total gained throughput; (b) number of path hops per VL; and (c) average time per VL embedding, i.e., VL path computation (mapping) and its consequent allocation.

random $< src, dst >$ pairs of hosts, where for each pair of endpoints we allocate as many virtual links as possible.

Experiment metrics. For each virtual link request we again measure the total gained throughput and the virtual link path hop count. In addition, we measure the time required to compute a path (virtual link mapping), and the time to allocate the computed path ($i.e.$, set appropriate flow rules within OpenFlow switches along the computed path). Note that the overall time for end-to-end virtual link embedding includes both virtual link mapping and its allocation. Our experiment goals are twofold: first, we want confirm our simulation results in real settings; secondly, we want to estimate an overhead of addressing constrained shortest path problem in real settings.

NM gains are confirmed experimentally. Using real-world settings, we were able to confirm constrained shortest path algorithms ($i.e.$, IBF, NM and EBFS) for the online TE produce superior performance even on a small physical network scale. This is similar to superior results of the offline TE which utilizes the constrained shortest path algorithms (see Section VII-B). Specifically, IBF, NM and EBFS show gains of up to 12% in total VL throughput (network utilization) and find almost 1 hop shorter VL path in average, w.r.t. extended Dijkstra (ED) shortest path scheme as shown in Figures 9a and 9b. Note however that IBF is applicable only for in l and $l \oplus 1$ cases (see Table I).

NM running time scales well with physical network size. Figure 9c shows how VL mapping is an order of magnitude faster for small scale physical networks (of $\approx 10^1$ nodes) than its allocation for all routing schemes that have been implemented. This is because the path computation is a local (in-memory) operation but the virtual link allocation requires setting up of flow rules within all switches along the loop-free underlying physical path found. Hence, its speed depends on the Round Trip Time between switches and the OpenFlow controller. In reference to Figures 8b and 8d, we can see how for large scale networks ($\geq 10K$ nodes), the running time of classical constrained shortest path algorithms such as EBFS can become prohibitive (up to two orders of magnitudes larger than in a case of a single path computation for small scale networks). As a result, the VL mapping time becomes a bottleneck. On the contrary, NM is just an order of magnitude slower at large-scale than at small scale. Thus, NM does not bottleneck the end-to-end VL embedding at large scale.

VIII. Conclusion

In this paper, we motivated the problem of achieving a *flexible* and *scalable* constrained shortest path management approach for virtual network services deployed across multiple

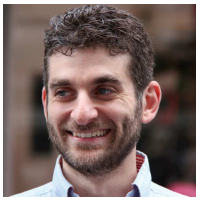
data centers. To cope with constrained shortest paths NP-hardness (that limits both its scalability and flexibility), we first introduced a novel algorithm viz., 'Neighborhoods Method' (NM), which utilizes a double pass (a synergy of dynamic programming and a branch-and-bound exhaustive search) and "Look Back" search space reduction techniques. Our computational complexity analysis indicates NM's quadratically lower complexity upper-bound than for recent branch-and-bound exhaustive search methods, and our scalability evaluation results show how NM is faster by an order of magnitude than these methods. Thus, NM *scales* better for large scale networks of $\geq 10,000$ nodes. Via numerical simulations of diverse network topology scenarios, we were able to show how the constrained shortest path management improves the network utilization (gains were seen up to 50%) and in some cases even in the energy efficiency of the recent management plane algorithms for virtual network embedding or network function virtualization service chaining. Additionally, we found improvements in the recent data plane algorithms for traffic engineering. Thus, we demonstrated that our proposed NM is also *flexible* and can be applied to diverse virtual network service scenarios. Finally, we were able to reproduce our main simulation results in a real-world GENI testbed with an NM implementation, whose source code is publicly available under a GNU license at [23].

As part of future work, we aim to develop new VNE/NFV-SC algorithms that can better utilize our proposed constrained shortest path scheme at larger network scales. To obtain a good lower bound solution within large network scale simulations, we can relax integrality constraints of the optimal VNE/NFV-SC integer programs to $e.g.$, use efficient linear programming.

REFERENCES

[1] VMware Cloud vServices - http://www.vmware.com/cloud-services.html; Last accessed Aug. 2018.
[2] M. Berman, et. al., "GENI: A federated testbed for innovative network experiments", *Elsevier ComNet*, 2014.
[3] F. Esposito, et al., "Slice Embedding Solutions for Distributed Service Architectures", *ACM Comput. Surv.*, 2013.
[4] A. Fischer, et al., "Virtual Network Embedding: A Survey", *IEEE CST*, 2013.
[5] F. Esposito, "CATENA: A Distributed Architecture for Robust Service Function Chain Instantiation with Guarantees", *IEEE NetSoft*, 2017.
[6] J. G. Herrera, J. F. Botero, "Resource allocation in NFV: A comprehensive survey", *IEEE TNSM*, 2016.
[7] A. Mendiola, et al., "A survey on the contributions of Software-Defined Networking to Traffic Engineering", *IEEE CST*, 2016.
[8] L. Lozano, A. L. Medaglia, "On an exact method for the constrained shortest path problem", *Elsevier COR*, 2013.
[9] X. Chen, et al., "Multi-criteria Routing in Networks with Path Choices", *IEEE ICNP*, 2015.
[10] P. Van Mieghem, F. Kuipers, "Concepts of exact QoS routing algorithms", *IEEE/ACM ToN*, 2004.
[11] R. G. Garroppo, et al., "A survey on multi-constrained optimal path computation: Exact and approximate algorithms", *Elsevier ComNet*, 2010.
[12] K. Nahrstedt, S. Chen, "Coexistence of QoS and best-effort flows-routing and scheduling", *IEEE TIW*, 1998.
[13] X. Yuan, X. Liu, "Heuristic Algorithms for Multi-constrained Quality of Service Routing", *IEEE INFOCOM*, 2001.
[14] J. M. Jaffe, "Algorithms for Finding paths with Multiple Constraints", *IEEE Network*, 1984.
[15] C. Y. Hong, et al., "Achieving high utilization with software-driven WAN", *ACM SIGCOMM CCR*, 2013.
[16] S. Jain, et al., "B4: Experience with a globally-deployed software defined WAN", *ACM SIGCOMM CCR*, 2013.
[17] R. Mijumbi, et al., "A path generation approach to embedding of virtual networks", *IEEE TNSM*, 2015.
[18] Y. Wang, et al., "A branch-and-price framework for optimal virtual network embedding", *Elsevier ComNet*, 2016.
[19] M. Chowdhury, et al., "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping", *IEEE/ACM ToN*, 2012.
[20] Z. Wang, J. Crowcroft, "QoS Routing for Supporting Resource Reservation", *IEEE JSAC*, 1996.
[21] F. Esposito, et al., "On Distributed Virtual Network Embedding with Guarantees", *ACM/IEEE ToN*, 2014.
[22] E. Danna, et al., "A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering", *IEEE INFOCOM*, 2012.
[23] Neighborhood Method Prototype - https://bitbucket.org/naas_cloud_computing_project/nm-of-controller/overview; Last accessed Aug. 2018.
[24] P. Khadivi, et al., "Multi-constraint QoS routing using a new single mixed metrics", *Elsevier JNCA*, 2008.
[25] A. Juttner, et. al., "Lagrange Relaxation based Method for the QoS Routing Problem", *IEEE INFOCOM*, 2001.
[26] R. Widyono, "The Design and Evaluation of Routing Algorithms for Real-time Channels", *ICSI, UC Berkeley*, 1994.
[27] M. S. Bazaraa, et al., "Linear programming and network flows", *John Wiley & Sons*, 2011.
[28] M. Chiesa, et al., "Lying Your Way to Better Traffic Engineering", *ACM CONEXT*, 2016.
[29] S. Vissicchio, et al., "Central control over distributed routing", *ACM SIGCOMM CCR*, 2015.
[30] D. Nace, et al., "Max-min fairness in multi-commodity flows", *Elsevier COR*, 2008.
[31] D. Eppstein, "Finding the k shortest paths", *SIAM SICOMP*, 1998.
[32] E. Amaldi, et al., "On the computational complexity of the virtual network embedding problem", *Elsevier ENDM*, 2016.
[33] M. Yu, et al., "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration", *ACM SIGCOMM CCR*, 2008.
[34] Y. Zhu, et al., "Algorithms for Assigning Substrate Network Resources to Virtual Network Components", *IEEE INFOCOM*, 2006.
[35] J. Londono, A. Bestavros, "Netembed: A Network Resource Mapping Service for Distributed Applications", *IPDPS*, 2008.
[36] J. Lischka, H. Karl, "A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection", *ACM VISA*, 2009.
[37] F. Samuel, et al., "PolyViNE: Policy-based Virtual Network Embedding across Multiple Domains", *Springer JISA*, 2013.
[38] M. T. Beck, et al., "Distributed and scalable embedding of virtual networks", *Elsevier JNCA*, 2015.
[39] R. Bellman, "On a Routing Problem", *AMS QAM*, 1958.
[40] D. Chemodanov, et. al., "A General Constrained Shortest Path Approach for Virtual Path Embedding", *IEEE LANMAN*, 2016.
[41] D. Chemodanov, et al., "A Constrained Shortest Path Scheme for Virtual Network Service Management", *arXiv:1604.08274*, 2018.
[42] R. Korf, "Depth-first Iterative-deepening: An Optimal Admissible Tree Search", *Elsevier AI*, 1985.
[43] Floodlight SDN controller - http://www.projectfloodlight.org; Last accessed Aug. 2018.
[44] N. McKeown, et al., "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM CCR*, 2008.
[45] A. Medina, et al., "BRITE: An Approach to Universal Topology Generation", *IEEE MASCOTS*, 2001.
[46] B. Waxman, "Routing of Multipoint Connections", *IEEE JSAC*, 1988.
[47] M. Faloutsos, et al., "On Power-Law Relationships of the Internet Topology", *ACM SIGCOMM CCR*, 1999.
[48] B. White, et. al., "An Integrated Experimental Environment for Distributed Systems and Networks", *USENIX OSDI*, 2002.
[49] A. P. Bianzino, et al., "Energy-aware routing: a reality check", *IEEE GLOBECOM*, 2010.
[50] IBM CPLEX solver - http://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer; Last accessed Aug. 2018.
[51] Open vSwitch - http://openvswitch.org; Last accessed Aug. 2018.

Dmitrii Chemodanov received his MS degree from the Department of Computer Science at Samara State Aerospace University, Russia in 2014. He is currently a PhD student in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, network and service management, and peer-to-peer networks.



Flavio Esposito is an Assistant Professor in the Computer Science Department at SLU and a Visiting Research Assistant Professor in the EECS Department at University of Missouri-Columbia. He received his Ph.D. in CS at Boston University in 2013, and his MS in Telecommunication Engineering from University of Florence, Italy. His research interests include network management, network virtualization and distributed systems.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Associate Professor in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, computer networking, and cyber security. He is a Senior Member of IEEE.

Andrei Sukhov received his PhD degree in Moscow, in Physics and Mathematics in 1993. In 2007 he received Dr.Sc. degree in computer networking at Moscow State University of Electronics and Mathematics (MIEM HSE). He is currently a Professor in the Department of Supercomputers and General Informatics at Samara National Research University. His current research interests include computer networks and security.