Optimal placement of protocol converters for connectivity: Complexity and algorithms

***Abstract*—Optimal placement problems are pervasive in networking: optimal placement of sensors, of content (caching), of proxies, of energy storage systems, etc. In the last years, most of the works focused on optimal placement of Virtual Network Functions (VNFs). They should be carefully placed in order to comply with some cost objectives. The recent works consider complex cases with several flows, capacity constraints, etc. Here, we consider a (seemingly) much simpler placement problem: ensuring connectivity through protocol converters placed on some nodes. Given a network with several types of nodes (*e.g.,* some using IPv4 and others IPv6), a link between two nodes of different types cannot be used unless one of its endpoints has a protocol converter. Our goal is to minimize the number of placed converters while ensuring the connectivity of the network. This problem has several use cases (*e.g.,* protocol interoperability, security, etc.).**

Despite the apparent simplicity of our problem, we prove that it is NP-complete and that under widely believed conjectures in complexity theory (i) it has no exact algorithm better than the brute-force one, i.e., no $2^{o(n+m)} \times poly(n)$ time exact algorithm, where n (resp. m) is the number of nodes (resp. of links) and $poly(n)$ any polynomial in n; and (ii) it cannot be approximated within a really better ratio than $\log n$. We design a provable performance greedy algorithm that matches this bound. Leveraging on the current efficiency of SAT and *Integer Linear programming* (ILP) solvers, we propose a SAT and ILP formulation of the problem and we compare the efficiency of both approaches. We perform extensive simulations on random and real topologies. It appears that the greedy algorithm is very fast and, surprisingly, gives almost always the exact solution. Considering exact algorithms, SAT and ILP solvers are complementary. While the latter is much faster on random topologies, the SAT solver outperforms the other approaches on real topologies.[1]

Keywords*—*Optimal placement; Complexity; Approximability; Connectivity ;* SAT *solvers; ILP solvers.

I. Introduction

Optimal placement is a general problem in networking. It consists in setting some components (software or hardware) in the network in order to ensure some proprieties under some constraints, while minimizing some cost. Several instances of this problem where considered in the literature. For example, there is an important work on sensor placement. Most of these works focus on the optimal placement of sensors in a two-dimensional space. The goal may be minimizing energy consumption [1], covering the maximum area for visual sensors [2], optimizing routing and flow assignments in the context of health monitoring [3], etc. Another example is optimal caching, mostly for streaming purpose [4], [5]. Recent works use machine learning techniques (see [6] for a recent survey on this subject). One can also cite the problem of proxy placement for optimizing traffic management[7] or security [8] purposes. However, the most active area dealing with optimal placement is *Network Function Virtualization* (NFV), which is a paradigm that consists of replacing hardware devices by software modules on standard servers. Such modules, also called middleboxes, or *Virtual Network Functions* (VNFs), should be carefully placed in the network in order to comply with some objectives: maximizing the amount of processed traffic, minimizing the delay, ensuring security, etc. The problem of optimal VNF placement received a lot of attention recently. Most of the works about this problem consider very complex cases: constraints on node and link capacity, or even multi-dimensional capacity constraints [9], several types of VNFs to process several types of flows, etc. Almost all these cases involve the resolution of NP-hard problems, even if we consider restricted topologies (*e.g.,* tree structured networks [10]).

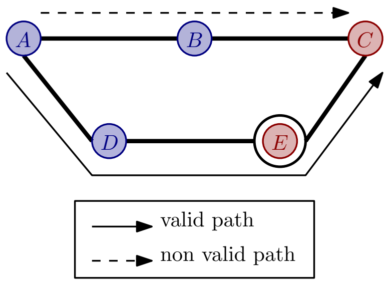


Fig. 1: Example of a network with two different type of nodes, and with a valid and a non valid path.

Here, we consider a (seemingly) much simpler case of optimal placement problem: ensuring connectivity. We deal with the following scenario: given a network with two different types of nodes (*e.g.,* IPv4 and IPv6), a link between two nodes of different types cannot be used unless a *converter* is placed on one of its endpoints. We relax all the capacity constraints. We want to make the network *really* connected, i.e., there is at least one path between each pair of nodes such that all the links in this path can be used (crossed). The problem we aim to solve is to minimize the number of placed converters (i.e., the number of nodes where a converter is deployed) in order to minimize management and deployment costs. We call this problem Bi-Con for *Bicolored Connectivity*. Figure 1 illustrates such a scenario. Here, there are two types of nodes (red and blue), and a converter is placed on E. The path $A-B-C$ is not valid since the link BC connects two nodes of different types. However, the path $A-D-E-C$ is valid, because the link DE can be used since E has a converter. This scenario has several application cases: ensuring connectivity in an

[1]Lamine: Il faudra raccourcir cet version pour que ça match avec les 200 mots d'EDAS. Mais la garder en backup pour l'intro.

IPv4/IPv6 network by connecting all the isolated IPv4 (or IPv6) islands by virtualized NAT64; deploying Deep Packet Inspection (DPI), firewalls or intrusion detection functions in order to protect some subareas of a network; or even on the physical layer: where to place wavelength converters, etc. The motivations and use cases are discussed in Section II-A. It appears that this problem is extremely difficult. In this paper, we prove several hardness results and we provide and analyze different algorithms to solve it. We perform extensive simulations in order to compare these algorithms.

Our contributions:

- We introduce and define formally Problem BI-CON and we describe several use cases;
- We prove several results about the hardness of the problem:
    - It is NP-complete;
    - Under a widely believed conjecture (the Exponential Time Hypothesis - ETH), there is no subexponential algorithm for the problem, i.e., there is no $2^{o(n+m)} \times poly(n)$ time algorithm to solve it, where n is the number of nodes, m the number of links and $poly(n)$ any fixed polynomial in n – and thus in m). This means that there is no really better algorithm than the brute force one unless ETH fails;
    - It is log-APX-hard, i.e., there is no polynomial algorithm with approximation ratio$(1-\epsilon)\log n$ for any $\epsilon > 0$ unless $\mathsf{P} = \mathsf{NP}$;
- We design a simple algorithm that drastically reduces (compresses) the instance size while preserving the optimal solution;
- We design a greedy algorithm to solve the problem and we prove that its approximation ratio is bounded by $\log(n-1) - \log OPT + 1$ (where OPT is the size of the optimal solution). This is asymptotically tight according to the log-APX-hardness of the problem;
- Leveraging on the efficiency of SAT and *Integer Linear Programming* (IPL) solvers in practice, we propose a model of our problem as a SAT formula to satisfy and as a binary ILP to solve;
- We compare all the aforementioned algorithms (greedy with approximation guarantee, brute force, using SAT and ILP solvers) through extensive simulations on random and real topologies. It appears that the greedy algorithm is extremely fast and that it finds almost always the optimal solution. For an exact resolution of the problem, the binary ILP approach is much faster on random topologies, but, surprisingly, using SAT solvers outperforms the other approaches on real topologies.

The paper is organized as follows: Section II describes the problem and its motivation, and discusses related work. Section III presents the model used in this paper and describes formally the problem. In Section IV, we prove several hardness results about its complexity and approximability. Section V presents a greedy approximation algorithm that meets the approximability hardness bound. The section also briefly describes the brute force algorithm. In Section VI, we propose a formulation (reduction) of the problem as a SAT formula and a binary ILP formulation. Section VII presents the results of extensive simulations of our different algorithms on random and real topologies. Finally, Section VIII concludes the paper and discusses some possible future work.

II. Problem, motivation and related work

A. Problem and motivation

We focus on a scenario where a network has two types of nodes. A link between two nodes cannot be used unless one of its endpoint has a converter. Each node in the network should be able to communicate with any other one. To do so, there must be at least one path between each pair of nodes where all the links can be used (we refer to it as a *valid* path). In such a path, either a link connects two nodes of the same type (we refer to such a link as *homogeneous*) and can be used directly, or a link connects two nodes of different types (we refer to such a link as *heterogeneous*) and at least one of its endpoints has a converter. In that case, we say that this link is *activated*. As in several other works, we aim to minimize the number of placed converter, (i.e., the number of nodes having a converter) in order to minimize deployment and management costs. Note that all the capacity constraints are relaxed. We focus only on connectivity. To the best of our knowledge, this case of optimal placement problem has never been investigated yet.

Our initial motivation was protocol interoperability. With the emergence of purely IPv6 (sub)networks, several IPv4 and IPv6 islands appear. All the nodes in such a network should be able to communicate, and it is a natural question to ask where the converters should be placed, for example, in the case where a part of the network is upgraded to IPv6.

There is a link between our problem and VNF placement. The IPv4/IPv6 interoperability can be addressed in an NFV framework, for example via virtualized NAT64 [11], [12] or other dedicated systems [13], [14], [15]. However, the problem of connectivity arises in several other use cases: if we want to “isolate” a part of a network for security purposes, and to ensure that all traffic entering in this area should be checked by some VNFs performing Deep Packet Inspection, detection intrusion or deploying firewalls. Only links having a VNF at one of its endpoints can be used, and the others should be disabled.

B. Related Work

We focus on related work about VNF placement, since it is the closer problem to ours. In recent years, a lot of work has been done about optimal VNF placements. The authors of [16] aim to minimize the distance between clients and VNFs serving them, while also minimizing the setup cost of these VNFs. The authors give an approximation algorithm with constant factor guarantee. In [17], the authors aim to take into account the impact of VNFs on the amount of traffic in the optimal placement process. They also deal with load balancing issues. The authors of [18] deal with a scenario where ISPs gradually upgrade their networks to SDN, and propose algorithms to optimize the location of upgraded networks. In [19], the authors give the first provable performance guarantees (an algorithm with logarithmic approximation ratio) to the problem of joint placement and allocation of VNFs. Due to the hardness of the problem, some works focus on specific topologies. For example, the work in [10] provides a quasi-polynomial algorithm for optimal VNF placement in tree structured networks. The authors of [9] deal with the same problem, but with multi-dimensional capacity constraints (different types of resource constraints for each flow). Considering VNF chaining, the authors of [20] propose a model based on SLA negotiation in the context of content delivery. In [21], the authors investigate the problem of VNF chaining in a multicast context. There are other works that deal with service chain orchestration [22], [23]. More recently, the authors of [24] deal with service function chaining with dynamic traffic. They assume that the functions are already placed and they aim to optimize the traffic allocation. Most of these works propose algorithms based on *Linear Programming* (LP), Integer LP (ILP) and Mixed ILP (MILP). Here, we want to leverage on the efficiency of SAT solvers and propose a second approach for placement problems. We compare our approach to the classical ((M)I)LP one.

C. Relation of our problem with other VNF placement problems

Problem BI-CON seems much simpler than other VNF placement problems. Unlike other works, it does not take into account routing optimization [18], fully flow processing [10], [19], [25], [9], order constraint on chained VNFs [24], [25], [23], [21], [20], link and node capacity constraints [10], [17], [26], etc. There is no metric constraints, and the only requirement is (real) *connectivity*. However, it is more difficult regarding some other aspects. Here, there is no set of customers or providers, *each* pair of nodes requires a valid path in order to communicate. The set of converters along a path may be seen as VNF chaining, but the number of converters in the path is not fixed and not known. These constraints make the problem more complex. In Section IV, we will see that, despite its apparent simplicity, Problem BI-CON is computationally hard.

III. MODEL AND PROBLEM FORMALIZATION

A. Model

We consider a network modeled as a simple connected undirected graph $G = (V, E)$ where $|V| = n$ is the number of nodes and $|E| = m$ is the number of links. For a node $u \in V$, we denote by $N(u)$ the open neighborhood of u, that is the set of nodes adjacent to u. We denote by $N[u]$ the closed neighborhood of u, i.e., $N[u] = N(u) \cup \{u\}$. Given a subset of nodes $S \subseteq V$, $N[S] = \bigcup_{u \in S} N[u]$.

We consider two types of nodes: a and b (for example IPv4 and IPv6). We express these types by a *color* function $c : V \to \{a, b\}$ where $c(u) = a$ (resp. b) if u is of type a (resp. b). Note that each node belongs to only one type, forming a partition of V, however, the graph is not necessarily bipartite, and the coloring is not proper. Such graphs are generally called *bicolored graphs* [27]. Thus, even if there exists a link $uv \in E$, nodes u and v cannot directly communicate if $c(u) \neq c(v)$ since they do not support the same communication protocol. In order to make the network *really* connected, and thus allow all the nodes to communicate, we have to put converters on some nodes. A converter is a software or hardware able to convert any packet from protocol a (resp. b) into a packet of the other protocol (*e.g.,* Dual Stack). In order to formalize the problem, we need some definitions.

Notation	Definition
$G = (V, E)$	A simply connected undirected graph
Bicolored graph	See Section III-A
$G^\star = (V^\star, E^\star)$	Compressed graph (see Section V-A)
$Conv$	The set of nodes with a converter
$n = \|V\|$ (resp. $\|V^\star\|$)	Number of nodes of G (resp. $G^\star$)
$m = \|E\|$ (resp. $\|E^\star\|$)	Number of links of G (resp. $G^\star$)
$N(u)$	Open neighborhood of node u
$N[u]$	Closed neighborhood of node u , i.e., $N(u) \cup \{u\}$
$N[S]$	Closed neighborhood of all nodes in $S \subseteq V$
$c(u)$	Color (i.e., type) of node u
H_t	Set of heterogeneous links
H_m	Set of homogeneous links
$uv \in H_t$	$uv \in E$ such that $c(u) \neq c(v)$
$uv \in H_m$	$uv \in E$ such that $c(u) = c(v)$
Valid path	See Definition 2
Really connected	See Definition 3
cc	Number of really connected components
$d^*(u)$	Number of different really connected components adjacent to node u
Candidates nodes	Nodes that are adjacent to an heterogeneous link
Activated link	A link uv such that u or v receives a converter (i.e., u or $v \in Conv$)

TABLE I: Summary of definitions and notations used.

Definition 1. *We consider two types of links:*

- *a link uv is homogeneous if $c(u) = c(v)$;*
- *a link uv heterogeneous if $c(u) \neq c(v)$.*

The set of homogeneous links is denoted by $H_m \subseteq E$, and the set of heterogeneous links is denoted by $H_t \subseteq E$. Let $Conv \subseteq V$ be the set of nodes that have a converter.

Definition 2. *A path $u_0u_1u_2 \ldots u_{\ell-1}u_\ell$ (with $u_i \in V$ and $0 \leq i \leq \ell$) is a valid path if and only if for each heterogeneous link u_iu_{i+1} ($0 \leq i \leq \ell - 1$) in the path, $u_i \in Conv$ or $u_{i+1} \in Conv$ (or both). We say that such a link is activated.*

It is clear that a valid path allows its endpoints to communicate, since at each crossed heterogeneous link, one of its endpoints has a converter.

Definition 3. *A bicolored graph $G = (V, E)$ is really connected if there is a valid path between each pair of nodes in V. A really connected component of G is a maximal subgraph of G that is really connected.*

If no converter is placed yet, the really connected components of G are the connected components of $G(V, E \setminus H_t)$, or equivalently $G(V, H_m)$, i.e., the connected components in the partial graph where the heterogeneous links are removed. Thus, the nodes of any really connected component are of the same type (color). The number of really connected components of G is denoted by cc.

In order to minimize the deployment and configuration costs, we aim to minimize the number of placed converters, and thus the number of nodes in $Conv$ such that each pair of nodes can communicate, i.e., there is a valid path between any pair of nodes in G. We call this problem BI-CON for *Bicolored Connectivity*.

Problem. BI-CON

$\min |Conv|$

s. t. the bicolored graph $G = (V, E)$ is really connected.

The decision version of Problem BI-CON is: given a bicolored graph $G = (V, E)$ and an integer k, is there a set $Conv \subseteq V$ of at most k nodes that makes G really connected? It is clear that a converter on a node that is only incident to homogeneous links is useless. The nodes incident to an heterogeneous link are called *candidate nodes*, since they are candidates to receive a converter. If we consider the set of really connected components as "nodes", the graph G is really connected if there is a tree which spans these "nodes". This tree has only activated links. Hence, any optimal solution requires $cc - 1$ activated links to really connect the graph. Note that $cc \leq n$, and that this bound is tight in the case of an alternating path (odd nodes and even nodes have different types). In this case, each really connected component contains only one node. More generally, we have the following inequalities:

$$0 \leq optimal\ solution \leq cc - 1 = activated\ links < n \quad (1)$$

Remark. Note that the model and almost all the algorithms presented in this paper can be easily generalized to any number of node types (colors), assuming that the converters allow any type to communicate with any other one.

IV. COMPLEXITY AND APPROXIMABILITY OF PROBLEM BI-CON

In this section, we give several hardness results about Problem BI-CON: there is no really better exact algorithm than the brute force one, and it has no polynomial algorithm that guarantees better than a logarithmic approximation ratio (under widely believed conjectures in complexity theory).

A. NP-*completeness of* BI-CON

In this section, we give a polynomial (and even linear) time reduction from the DOMINATING SET problem to the decision version of Problem BI-CON. Recall that, given a graph $G = (V, E)$, a *dominating set* of G is a subset of nodes $D \subseteq V$ such that each node not in D has at least one neighbor in D. In other words, $N[D] = V$. Given a graph G and an integer k, the decision version of the DOMINATING SET problem asks whether G admits a dominating set of size at most k or not. It is well known that DOMINATING SET is NP-complete. To prove

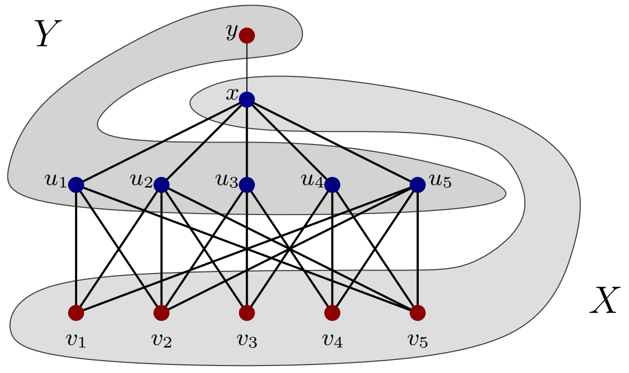


Fig. 2: A Roof graph and its reduction to a bicolored graph. Here the bipartition is $X = \{x\} \cup \{v_1, \ldots, v_5\}$ and $Y = \{y\} \cup \{u_1, \ldots, u_5\}$. The colors set after the reduction are *blue* and *red*.

the NP-completeness of the decision version of Problem BI-CON, we need to define a special class of graphs that we name *Roof graphs*[2]. See Figure 2 for an illustration.

Definition 4. *A bipartite graph $G = (X \cup Y, E)$ is a Roof graph if and only if it satisfies the following conditions:*

- *(i)* $|X| = |Y|$;
- *(ii) There exists $x \in X$ such that for every $v \in Y$, the link $xv \in E$;*
- *(iii) there exists a node $y \in Y$ such that the only neighbor of y is x, i.e., $N(y) = \{x\}$.*

The authors of [28] prove that DOMINATING SET is NP-complete even when restricted to bipartite graphs. They use a reduction from any graph to a Roof graph. Thus, DOMINATING SET is NP-complete even when restricted to Roof graphs.

Proposition 1. *The decision version of Problem* BI-CON *is* NP-*complete.*

Proof: It is trivial that the decision version of Problem BI-CON is in NP. Thus we will prove its NP-hardness.

Let $G = (X \cup Y, E)$ be a Roof graph. Let $Y' = (Y \setminus \{y\}) \cup \{x\}$ and $X' = (X \setminus \{x\}) \cup \{y\}$. We set the coloring as follows: $c(u) = a$ for every node $u \in X'$, and $c(u) = b$ for every node $u \in Y'$ (see Figure 2 for an illustration). We prove that G has a dominating set D of size at most k if and only G has a solution $Conv$ of size at most k for Problem BI-CON.

($\Rightarrow$) Suppose that the Roof graph $G = (X \cup Y, E)$ has a dominating set D of size at most k. First, observe that the subgraph induced by Y' is really connected, since all the nodes in Y' have the same color and that all the $u_i \in Y \setminus \{y\}$ are adjacent to x. Thus, no converter is needed to really connect the nodes of Y'. Observe also that, for each node $v \in X'$, all its neighbors are in Y', i.e., $\forall v\ X', N(v) \subseteq Y'$. Thus, in order to dominate each node $u \in X'$, node u or one of its neighbors (or both) is in D, i.e., $N[u] \cap D \neq \emptyset$.

[2]The term is ours. Because these graphs look like houses with an antenna on the roof (see Figure 2). The authors of [28], [29] do not use this terminology.

Let us set $Conv = D$. Then there is a valid path between any node in X' and any node in the really connected component induced by Y', since any $v \in X'$ or one of its neighbors has a converter. Thus, $Conv$ really connects G and $|Conv| = |D| \leq k$.

($\Leftarrow$) Suppose now that G has a set $Conv$ of size at most k that really connects it. We will show how to compute in linear time a dominating set D of G of size at most k.

- We start with $D \leftarrow \emptyset$;
- We know that $D \cap \{x, y\} \neq \emptyset$ in order to dominate y. We set $D \leftarrow D \cup \{x\}$;
- So now $\{x\} \cup Y$ are dominated (since all the nodes in Y are adjacent to x). Let us consider the nodes $v_i \in X' \setminus \{y\}$ (or equivalently $X \setminus \{x\}$). Recall that all their neighbors are in Y'. Thus, each v_i or one of its neighbors u has a converter (i.e., is in $Conv$), since $\forall u \in N(v_i), c(v_i) \neq c(u)$.

Thus, $D = (Conv \setminus \{y\}) \cup \{x\}$ is indeed a dominating set of G of size at most k (it cannot be of size $k+1$ since either x or y must be in $Conv$). ■

B. Exponential Time Hypothesis and exponential lower bound

Currently, there is no known (deterministic or randomized) algorithm for 3-SAT running in $2^{o(N)} \times poly(M)$, where N (resp. M) is the number of variables (resp. clauses) of the 3-SAT formula, and $poly(M)$ any fixed polynomial. The Exponential Time Hypothesis (ETH) [30] states that such an algorithm does not exist. ETH is widely believed to be true, and a lot of conditional lower bounds for other problems were derived from ETH. Violating ETH would be a breakthrough, since it would lead to a better (i.e., subexponential) algorithm not only for 3-SAT, but also for several other problems [31], including DOMINATING SET.

Proposition 2 ([31])**.** *There is no $2^{o(n+m)} \times poly(m)$ exact algorithm for* DOMINATING SET*, unless ETH fails.*

Note that if a problem A has a *linear* reduction to a problem B, and if an $2^{o(n+m)} \times poly(m)$ algorithm for problem A implies that ETH fails, then any $2^{o(n+m)} \times poly(m)$ algorithm for problem B would violate ETH. The reduction from DOMINATING SET to DOMINATING SET restricted to Roof graphs used in [28] is linear. Thus, any $2^{o(n+m)} \times poly(m)$ algorithm for the latter problem would violate ETH. Note that our reduction used in Section IV-A is also linear. It immediately follows that:

Proposition 3. *There is no $2^{o(n+m)} \times poly(m)$ exact algorithm for the decision version of Problem* BI-CON *unless ETH fails.*

This means that the exact brute force algorithm for BI-CON is roughly optimal. More precisely, the exponent in the exponential part of the complexity formula is asymptotically optimal unless ETH fails.

C. Approximability lower bound

The complexity class log-APX is the class of optimization problems that have a polynomial algorithm with $O(\log n)$ approximation ratio. An optimization problem is log-APX-hard if any problem in log-APX can be reduced[3] to it. The author of [33] show that the log-APX-hard problem DOMINATING SET has no polynomial algorithm with approximation ratio $(1 - \epsilon) \log n$ for any $\epsilon > 0$, unless P = NP. The authors of [29] show that the DOMINATING SET problem is log-APX-hard even when restricted to Roof graphs, and *via* the reduction in their proof, the aforementioned bound also applies to the problem. We prove that the optimization version of BI-CON has the same approximation bound.

Proposition 4. *There is no polynomial time algorithm for Problem* BI-CON *with approximation ratio $(1 - \epsilon) \log n$, for any $\epsilon > 0$, unless unless* P = NP.

Proof: Let $G = (V, E)$ be a Roof graph. We *color* it as in the reduction in Section IV-A, i.e., *color* a (blue) for $(Y \setminus \{y\}) \cup \{x\}$ and *color* b (red) for $(X \setminus \{x\}) \cup \{y\}$. Let OPT be the size of the optimal solution for DOMINATING SET (the size of a smallest dominating set D of G), and OPT' be the size of the optimal solution of BI-CON (the size of a smallest set $Conv$ that really connects G).

Recall that in the reduction in Section IV-A, we proved that for any dominating set D of G, the set $Conv = D$ is also a solution to Problem BI-CON. This implies that $OPT \geq OPT'$. On the other hand, we also showed that for any solution $Conv$ of Problem BI-CON the set $D = (Conv \setminus \{y\}) \cup \{x\}$ is a dominating set of G and $|D| \leq |Conv|$. This implies that $OPT \leq OPT'$, and thus that $OPT = OPT'$.

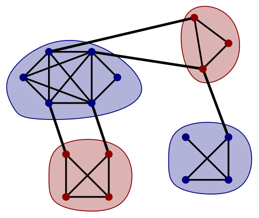
Suppose for the sake of contradiction that there is a polynomial-time algorithm for the optimization version of Problem BI-CON with approximation ratio $o(\log n)$. Such an algorithm would always return a solution $Conv$ of size at most $o(\log n) \cdot OPT'$. By setting $D = (Conv \setminus \{y\}) \cup \{x\}$ we get a dominating set of size at most $o(\log n) \cdot OPT' = o(\log n) \cdot OPT$ and thus a polynomial-time algorithm with approximation ratio of $o(\log n)$ for DOMINATING SET. ■

V. ALGORITHMS FOR MINIMIZING THE NUMBER OF CONVERTERS

A. Computing the compressed graph

As noted before, any optimal solution for Problem BI-CON does not involve placing a converter on a node that has the same color (i.e., type) as all its neighbors. Or, equivalently, on a node that is not adjacent to any heterogeneous link. Thus the graph of an instance of our problem can be converted into a smaller graph which has exactly the same optimal solutions. This compressed graph will be given as input of all the algorithms proposed in this paper. Let us define the set of *really connected components* as the set of (simply) connected components of the partial graph $G' = (V, E \setminus H_t)$, i.e., the set of connected components of the graph after deleting all the heterogeneous links. The compression of the graph can then be done by keeping only the candidate nodes and the heterogeneous links, and creating dummy links between

[3]Here, the definition of reduction is different from a classical polynomial reduction. It is a stronger form of reduction that preserves approximability. For sake of simplicity and to avoid too technical definitions, we do not define formally such a reduction. The interested reader can refer to [32] for more details.


(a) A bicolored graph $G = (V, E)$.

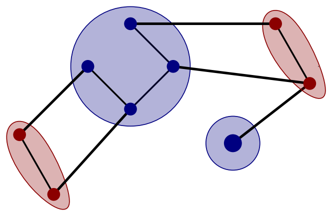

(b) The corresponding compressed graph $G^\star = (V^\star, E^\star)$.

Fig. 3: Example of graph compression.

nodes of the same really connected component. We compute a *compressed graph* $G^\star = (V^\star, E^\star)$ from a bicolored graph $G = (V, E)$ as follows:

- $V^\star \leftarrow$ the candidate nodes;
- All the heterogeneous links of E belong to $E^\star$, i.e., $H_t \subseteq E^\star$;
- The candidate nodes that belong to the same really connected component are linked by an homogeneous path.

Figure 3 gives an example of compression of a graph. The complexity of this compression is linear, i.e., in $O(n + m)$, since we need to (i) remove the heterogeneous links, (ii) compute the connected components of the remaining graph, (iii) create a path linking all the candidate nodes in the same connected component, and finally (iv) add the previously deleted heterogeneous links. All these operations are linear. In the rest of the paper, we will denote by n (resp. m) the number of nodes (resp. links) of the compressed graph $G^\star = (V^\star, E^\star)$.

B. The Greedy Algorithm

A simple and intuitive greedy algorithm would be to iteratively select the node adjacent to the largest number of different really connected components, and then to place a converter on it. Let cc be the number of really connected components of the compressed graph $G^\star = (V^\star, E^\star)$. When a link is activated between two really connected components, then these components are merged (since they can communicate) and form a new single really connected component. Thus, after each iteration, the added converter decreases the number of really connected components by at least one. Let us denote by cc_i the number of really connected components of $G^\star$ after the i^{th} iteration of the greedy algorithm. In particular, $cc_0 = cc$. The algorithm stops with a feasible solution when $cc_i = 1$, i.e., the network is really connected. Since the algorithm adds a node to $Conv$ at each iteration, the number of iterations is also the size of the solution given by the algorithm. Let us denote by $d_i^*(u)$ the number of different really connected components adjacent to node u after the i^{th} iteration of the algorithm. At each iteration, the algorithm selects the node u having the largest $d_i^*(u)$ to receive a converter.

Note that Algorithm 1 recomputes the really connected components at each iteration. This step is also used to update the values of $d_i^*(u)$ for all the nodes. Thus, each iteration is in $O(n + m)$. The number of iterations is bounded by $cc - 1 < n$. Hence, the complexity of Algorithm 1 is bounded by $O(n(n + m))$.

Algorithm 1 Greedy algorithm for Problem BI-CON

```
Input: A bicolored graph $G = (V, E)$
 1: Compute $G^\star = (V^\star, E^\star)$
 2: $i \leftarrow 0$
 3: $Conv \leftarrow \emptyset$
 4: while $cc_i > 1$ do
 5:     $u \leftarrow \max_{u \in V^\star} d_i^*(u)$
 6:     $Conv \leftarrow Conv \cup \{u\}$
 7:     Update the really connected components
 8:     Compute the values of $d_{i+1}^*(u)$ for each $u \in V^\star$
 9:     $i{+}{+}$
10: end while
11: Return $Conv$
```

1) Approximation ratio of Algorithm 1:

Here, we investigate the approximation ratio of Algorithm 1. We have the following result:

Proposition 5. *The approximation ratio of Algorithm 1 is bounded by*

$$\log(n - 1) - \log(OPT) + 1$$

where OPT is the size of the optimal solution.

Proof: Since, at each iteration, one node is added to the solution, we investigate a bound on the number of iterations of Algorithm 1 in order to derive its approximation ratio.

Let k_i be the optimal number of converters to add after the i^{th} iteration in order to make the graph really connected. Thus, $k_0 = OPT$. We know that we have to activate exactly $cc - 1$ heterogeneous links in order to really connect the network. Before the first iteration, there are $cc_0 - 1$ links to activate, and $k_0 = k$ nodes that must have a converter. So, the average number of different really connected components adjacent to each node of the optimal solution (i.e., the average value of $d_i^*(u)$ for all nodes u) is $\frac{cc_0 - 1}{k_0}$. By the pigeonhole principle, there must be a node u with $d_0^*(u) \geq \left\lceil \frac{cc_0 - 1}{k_0} \right\rceil$. The greedy algorithms adds the node u with the largest $d_0^*(u)$ to $Conv$, and thus $cc_1 = cc_0 - d_0^*(u)$. We get that

$$\begin{aligned} cc_1 &= cc_0 - \max_{u \in V^\star} d_0^*(u) \\ &\leq cc_0 - \left\lceil \frac{cc_0 - 1}{k_0} \right\rceil \\ &\leq cc_0 - \frac{cc_0 - 1}{k_0} \end{aligned} \tag{2}$$

Also by the pigeonhole principle, one can show by induction that

$$\forall i,\ cc_{i+1} \leq cc_i - \frac{cc_i - 1}{k_i} \tag{3}$$

We distinguish two phases in the analysis of this algorithm.

Phase 1. While $cc_i - 1 > k_i$, it is possible that the node added by Algorithm 1 does not belong to any optimal solution with k_i nodes. It is then possible that $k_{i+1} = k_i$. We get the following bound:

$$\forall i,\ cc_{i+1} \leq cc_i - \frac{cc_i - 1}{k} \tag{4}$$

It is sure that this phase ends when $cc_i - 1 \leq k$ (in fact when $cc_i - 1 = k$). Let us study the recursive sequence $u_{i+1} =$

$u_i - \frac{u_i - 1}{k}$, with $u_0 = cc_0$. This sequence can be rewritten as $u_{i+1} = \frac{k-1}{k} \cdot u_i + \frac{1}{k}$, which is an arithmetico-geometric sequence with a well-known closed form:

$$u_i = \left(\frac{k-1}{k}\right)^i (u_0 - 1) + \frac{1}{k} \qquad (5)$$

And thus $cc_i - 1 \leq u_i - 1 \leq k$ when

$$i \geq \frac{\log(cc_0 - 1) - \log(k - \frac{1}{k})}{\log k - \log(k-1)} . \qquad (6)$$

Phase 2. When $cc_i - 1 = k_i$ (recall that $\forall i,\ k_i \leq cc_i - 1$), then $cc_{i+1} = cc_i - 1$, it remains at most $k - 1$ iterations to get $cc_i = 1$. Thus, there are at most $\frac{\log(cc_0 - 1) - \log(k - \frac{1}{k})}{\log k - \log(k-1)}$ iterations until $cc_i - 1 = k_i$, then $k - 1$ iterations until $cc_i = 1$. The maximum number of iterations (and the maximum size of a solution returned by the greedy algorithm) is

$$i = \frac{\log(cc - 1) - \log(OPT - \frac{1}{OPT})}{\log OPT - \log(OPT - 1)} + OPT - 1 \qquad (7)$$

The approximation ratio (i.e., $solution/OPT$) is

$$i = \frac{\log(cc - 1) - \log(OPT - \frac{1}{OPT})}{OPT(\log OPT - \log(OPT - 1))} + \frac{OPT - 1}{OPT} \qquad (8)$$

Note that $(\log OPT - \log(OPT - 1)) = 1/OPT + o(1/OPT)$ and thus the denominator is equal to $1 + o(1)$. The approximation ratio is bounded by

$$\frac{\log(cc - 1) - \log(OPT - 1)}{1 + o(1)} + 1$$

Recall that $cc \leq n$. This concludes the proof. ■

C. *Exact algorithm: Branch & Bound*

An exact solution to Problem BI-CON can be found using a brute force strategy. According to Proposition 3, there is no asymptotically better algorithm (in the worst case) unless ETH fails. Due to space limitation, we omit the details about this algorithm which is a classical Branch & Bound algorithm that explores all possible solutions. In the worst case, it enumerates all the subsets of $V^\star$. Thus, it runs in time $O(2^n \times poly(m))$.

VI. USING SAT AND ILP SOLVERS

A. *Reductions to* SAT

1) Preliminaries: SAT is a central problem in computer science. It is the first problem that was proven to be NP-complete, and thus any problem in NP, like the decision version of problem BI-CON, can be polynomially reduced to it. SAT solvers are often used to solve industrial-strength problems involving millions of variables [34]. While we do not know any algorithm performing better than $O(2^N \times poly(N))$ (where N is the number of Boolean variables) in the worst case[4], SAT solvers perform very well in practice for reasons we do not yet fully understand.

[4]However, for special instances of SAT there are slightly better algorithms. For example 3-SAT has an $O(1.439^N)$ deterministic algorithm, and an $O(1.321^N)$ randomized algorithm. Note that this is not in contradiction with ETH, since $a^N = 2^{N \log_2 a}$

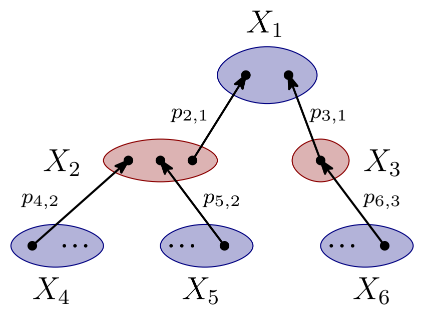


Fig. 4: A spanning tree over the really connected components to ensure connectivity.

Our approach is to reduce the decision version of Problem BI-CON to SAT and then to use SAT solvers to solve it, i.,e., from an instance $(G^\star, k)$ of Problem BI-CON, build a SAT formula $\phi_{G^\star,k}$ that is satisfiable if and only if $G^\star$ can be really connected using at most k converters.

To solve the optimization version, we just need to iterate the process. For example, take the size k of a solution returned by the greedy algorithm. Then decrement iteratively k until the formula $\phi_{G^\star,k}$ is not satisfiable. This means that the optimal solution is of size $k+1$. In addition to their good performances, the main motivation for using SAT solvers is that, unlike most of NFV placement problems, Problem BI-CON has no metric constraints (*e.g.,* node limited capacity, path length, amount of processed flows, etc.). We only focus on real connectivity in a broad sense. Thus, Boolean variables are sufficient in our case.

Recall that a SAT formula is a mathematical expression built from Boolean variables and logic operations: *AND* or *conjunction* denoted by $\wedge$, *OR* or *disjunction* denoted by $\vee$, *negation* denoted by $\neg$, *implication* denoted by $\implies$, etc.

2) SAT *reduction:* We propose a reduction to SAT that has the advantage to avoid the exploration of all possible valid paths, and thus involves fewer variables and shorter formulas. The real connectivity is ensured by building a spanning tree over the set of really connected components.

Let $G^\star = (V^\star, E^\star)$ be the compressed graph of the initial network $G = (V, E)$. Recall that we want to know if it is possible to put converters on k nodes in order to ensure real connectivity. Let n be the number of nodes in $G^\star$, and let $X_1, \ldots, X_{cc}$ be the really connected components of $G^\star$. We introduce the variable x_u for each $u \in V^\star$. This variable $x_u = true$ if and only if u has a converter, i.e., $u \in Conv$. To ensure real connectivity, we will check if the set of *activated* nodes spans the set of really connected components, i.e., a spanning tree where the "nodes" are the really connected components. To do so, we introduce the *parent relation* and a variable $p_{j,j'}$ for each pair of really connected components X_j and $X_{j'}$ such that $j \neq j'$. The variable $p_{j,j'} = true$ if and only if the really connected component $X_{j'}$ is the parent of X_j in the tree. This means that there is a link uv such that $u \in X_j$ and $v \in X_{j'}$, and $u \in Conv$ or $v \in Conv$. And thus X_j and $X_{j'}$ are really connected by an activated link. See Figure 4 for an illustration of this approach. Indeed, each really connected component has only one parent, except the root which has none. This constraints leads to the following formula

$$p_{j,j'} \implies \bigvee_{\substack{u\in X_j\\ v\in X_{j'}}} x_u \vee x_v \tag{9}$$

However, this is not enough to ensure real connectivity. It is possible to have a cycle if, for example, $X_{j''}$ is the parent of $X_{j'}$, $X_{j'}$ the parent of X_j, and X_j the parent of $X_{j''}$. Thus, we have to set a *level* to each component in the tree. We introduce the variable $\ell_{j,i}$ for each really connected component j and each level i. We have $\ell_{j,i} = true$ if and only if X_j is at level i in the spanning tree. Since the root is at level 0, the highest possible level is $cc-1$, in the case where the tree of really connected components is a path. This modifies the *parent* formula that we denote by $parent(j,j')$ to

$$p_{j,j'} \wedge \left[\left(\bigvee_{\substack{u\in X_j\\ v\in X_{j'}}} x_u \vee x_v\right) \wedge \left(\bigvee_{1\le i\le cc-1} \ell_{j,i} \wedge \ell_{j',i-1}\right)\right] \tag{10}$$

and the main formula is

$$\bigwedge_{1<j\le cc} \bigvee_{1\le j'\le cc} parent(j,j') \tag{11}$$

This formula is equivalent to an instance of Problem Bi-Con under some uniqueness and cardinality rules in order to ensure a bijection between the variables set to $true$ and the solution of Problem Bi-Con. For cardinality, we must

The uniqueness rule over a set of variable $\{x_1,\dots,x_k\}$ can easily be expressed by the formula

$$Uniq(\{x_1,\dots,x_k\}) = (\bigvee_{1\le i\le n} x_i) \wedge (\bigwedge_{1\le i<j\le n} \neg x_i \vee \neg x_j) \tag{12}$$

If the first part $(\bigvee_{1\le i\le n} x_i)$ is removed, then the formula expresses the rule *at most one* variable is $true$, which we denote by $AtMostOne(\{x_1,\dots,x_k\})$.

The uniqueness rules to ensure the equivalence between the instance of Problem Bi-Con and the SAT formula are

$$\forall j\in\{2,\dots,cc\} \qquad Uniq(p_{j,1},\dots,p_{j,cc}) \tag{13}$$
$$\forall j\in\{1,\dots,cc\} \qquad Uniq(\ell_{j,0},\dots,\ell_{j,cc-1}) \tag{14}$$
$$\forall u\in V^\star \qquad AtMostOne(x_{u,1},\dots,x_{u,k}) \tag{15}$$
$$\forall i\in\{1,\dots,k\} \qquad Unique(\cup_{u\in V^\star}\ x_{u,i}) \tag{16}$$

Formulas (13) and (14) ensure that each really connected component (except the root) has exactly one parent and belongs to only one level. Formulas (15) and (16) ensure that each node has at most one converter and each one is placed on only one node. These formulas can easily be translated into SAT subformulas. Formula $\phi_{G^\star,k}$ is satisfiable if and only if the corresponding instance of Problem Bi-Con is positive. Moreover, the variables $x_{u,i}$ which are $true$ give us the nodes where to place the converters.

Recall that the number of converters (k), the number of levels and the number of really connected components (cc) are all bounded by n. The number of variables is in $O(n^2)$. Each $Uniq$ (or $AtMostOne$) formula is in $O(n^2)$, and the number of $Uniq$ formulas created is in $O(n)$. Thus, the size of the Formulas (13) to (16) is in $O(n^3)$. Note that Formula (11) involves each link only k times. Its size is bounded by $O(n{\cdot}m)$. It follows that the size of the whole formula is in $O(n^3)$, where n is the number of nodes of $G^\star$.

B. *Binary ILP formulation of Problem* Bi-Con

In order to compare the efficiency of the SAT approach and an ILP approach, we designed a binary ILP formulation of Problem Bi-Con based on the same method as the SAT reduction. Unfortunately, due to the limited space, we cannot detail it. The binary variables used are roughly the same as the Boolean variables in SAT: $x_u = 1$ for each node receiving a converter, $z_{uv} = 1$ for each activated link, $p_{j,j'} = 1$ if $X_{j'}$ is parent of X_j, etc. The AND and OR operations can easily be translated into ILP constraints. For example, expressing that either u *or* v has an NVF can be written as

$$\begin{aligned} z_{u,v} &\ge x_u\\ z_{u,v} &\ge x_v\\ z_{u,v} &\le x_u + x_v \end{aligned} \tag{17}$$

This ensures that $z_{uv} = 1$ if and only if $x_u = 1$ or $x_v = 1$. The total size of the binary ILP is in $O(n^3)$. With the same approach, we get roughly the same formula size for SAT and for ILP.

VII. Simulations

A. *Implemented algorithms*

We implemented the different algorithms presented in this paper. The compression, greedy and brute force algorithms are implemented in Python 3 using the NetworkX package [5]. We use the PuLP package[6] to formulate the binary ILPs corresponding to the different topologies, and then CPLEX to solve them.

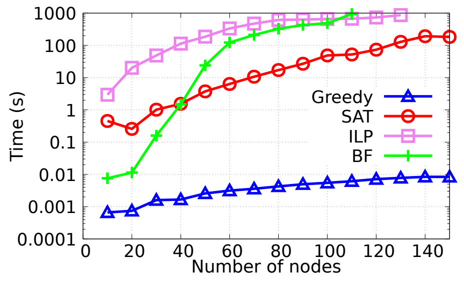
To solve the SAT formulas, we wanted to use Glucose [35][7] which is one of the most efficient solvers. However, Glucose processes only formulas under *Conjunctive Normal Form* (CNF), which are conjunctions of disjunctions. Our SAT formulas are not in CNF. The Z3 Theorem Prover [8] offers tools to write SAT formulas and convert them into CNF. Thus, we use it for this purpose. We then run Glucose to solve the formulas under CNF obtained thanks to Z3.

All the simulations are performed on devices with an Intel Xeon Gold 6130 2.1 Ghz processor.

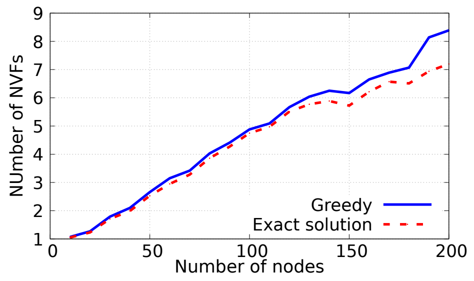
B. *Simulations on random topologies*

We aim to generate very sparse random graphs, because the sparser is the graph, the more difficult is the problem. Indeed, we observed that ? if the graph is too dense, there are almost always only two really connected components, one of each type. And a single converter is enough to really connect them. Thus, we generate power law graphs according to the Barabási--Albert model [36]. This model is closer to real topologies

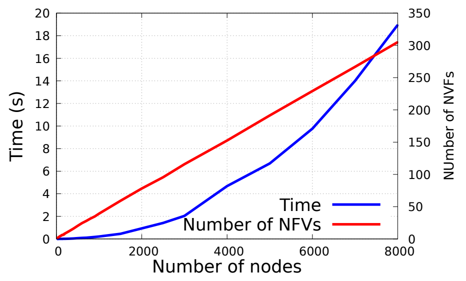
[5]networkx.github.io/
[6]pypi.org/project/PuLP/
[7]https://www.labri.fr/perso/lsimon/glucose/
[8]github.com/Z3Prover/z3



(a) Processing time according to the network size. Note the logarithmic scale on the y-axis.

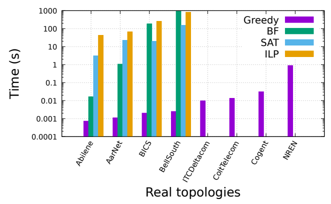


(b) Solution of the greedy algorithm vs. exact solution.

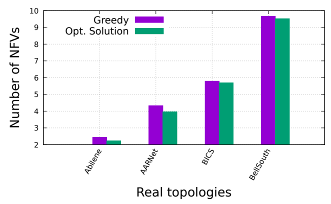


(c) processing time and solutions of the greedy algorithm.

Fig. 5: Results on random topologies.



(a) Processing time. Note the logarithmic scale on the y-axis.



(b) Solution of the greedy algorithm vs. exact solution.

Fig. 6: Results on random topologies.

than the Erdős-Rényi model. The graphs are generated with parameter 3, i.e., $m \approx 3n$. For each graph size, we generate 100 instances where the types are randomly set on the nodes (with probability $1/2$ for each type). The results are averaged over the 100 instances.

Figure 5a shows the processing time of the different algorithms. Recall that all the algorithms are exact ones except the greedy algorithm. Obviously, the greedy algorithm is the fastest one. It takes less than one second for 150 nodes. It appears that the SAT approach is much faster than the ILP and brute force ones. For example, for 100 nodes, the greedy algorithm takes 0.005s., the SAT approach takes approximately 48s., while the brute force algorithm takes 487s. and the ILP approach takes 657s. However, after this size, it seems that the ILP is faster than the brute force. Figure 5b shows the solutions (i.e., number of converters placed) of the greedy algorithm compared to the exact solution provided by the other algorithms. The approximate solution is very close to the exact one. For example, for 100 nodes, the greedy algorithm places on average 4.88 converters, while the average exact solution is 4.75. For 200 nodes, the greedy algorithm places on average 8.39 converters, while the average exact solution is 7.2. Moreover, it scales very well. Figure 5c shows its processing time over large graphs. It takes 0.2s., for 1000 nodes, less than 1s. for 2000 nodes, and 6.69s. for 5000 nodes. The tradeoff between speed and efficiency is clearly the best with the greedy algorithm. However, for an exact solution, its seems that the SAT approach outperforms the other ones.

Topology	Number of nodes	Number of links	Year
Abilene	11	14	2005
AARNet	19	24	2010
BICS	33	48	2011
Bell South	51	66	<2005
ITC Deltacom	113	161	2010
Colt Telecom	153	177	2010
Cogent	197	243	2010
NREN	1157	1465	2011

TABLE II: Real topologies used for simulations [37].

C. *Simulations on real topologies*

We also simulated our algorithms on several real topologies. They are described in Table II. All these topologies are taken from the data set “Topology Zoo” [37][9]. The number of links may differ from the original data since we removed parallel links. We generate 100 instances of each topology where the type of each node is randomly set. The results are averaged over these instances.

The exact algorithms do not take reasonable time for the last four topologies. Thus, we simulated them only on the first four. Figure 6a shows the processing time of the different algorithms. The greedy algorithm is extremely fast, it takes 0.0007s. for Abilene, and 0.8s for NREN. Among the exact algorithms, the SAT approach is the fastest one. It takes 151s. on average for the Bell South topology, while the brute force algorithm takes 918s. and the ILP 821s.

Figure 6b shows the average number of converters placed by the greedy algorithm compared to the optimal solution. As we could not obtain the exact solutions for the last four topologies, we only compare the results for the first ones. The results of the greedy algorithm are very close to the optimal solution. For example, the average optimal solution for the Bell South topology is 9.52s, while the solution obtained by the greedy algorithm is 9.67s on average.

VIII. Conclusion and Future Work

Optimal placement of VNFs is currently an important challenge in networking. In this paper, we investigated a seemingly simple scenario of optimal NVF placement where some links must have converter at one of their endpoints

[9]http://www.topology-zoo.org

in order to be used. The goal is to minimize the number of placed converters while ensuring the connectivity of the network. This scenario may appear in several use cases, such as protocol interoperability, security, etc. We proved that this problem is actually very hard. It is NP-complete and, under some widely believed conjectures in complexity theory, it has no really better algorithm than the brute force one, and it cannot approximated within less than a logarithmic factor. We designed and analyzed a greedy algorithm that meets this approximability bound. We also provided a SAT and an ILP formulation of the problem, and we showed through extensive simulations that using a SAT solver is much faster than using an ILP one. This shows that using SAT solvers is a promising approach for networking optimization problems. For its part, the greedy algorithm performs very well in practice, giving almost optimal solutions, while it can solve instances of thousands of nodes in a few seconds.

As a future work, we plan to adapt the proposed algorithms to more constrained scenarios with limited capacities on the nodes and the links, and several types of converters. Another future work would be the optimal placement of tunnel endpoints in order to ensure communication in multilayer networks.

References

[1] P. Cheng, C.-N. Chuah, and X. Liu, "Energy-aware node placement in wireless sensor networks," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, vol. 5. IEEE, 2004, pp. 3210–3214.

[2] E. Hörster and R. Lienhart, "On the optimal placement of multiple visual sensors," in *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, 2006, pp. 111–120.

[3] M. Elsersy, T. M. Elfouly, and M. H. Ahmed, "Joint optimal placement, routing, and flow assignment in wireless sensor networks for structural health monitoring," *IEEE Sensors Journal*, vol. 16, no. 12, pp. 5095–5106, 2016.

[4] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "Qoe-driven mobile edge caching placement for adaptive video streaming," *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2017.

[5] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and X. Tao, "Distributed online learning of cooperative caching in edge cloud," *IEEE Transactions on Mobile Computing*, 2020.

[6] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey," *Journal of Network and Computer Applications*, p. 103005, 2021.

[7] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 3. IEEE, 1999, pp. 1282–1290.

[8] J. Cesareo, J. Karlin, J. Rexford, and M. Schapira, "Optimizing the placement of implicit proxies," 2012.

[9] G. Sallam, Z. Zheng, and B. Ji, "Placement and allocation of virtual network functions: Multi-dimensional case," in *IEEE ICNP*, 2019, pp. 1–11.

[10] Y. Chen, J. Wu, and B. Ji, "Virtual network function deployment in tree-structured networks," in *IEEE ICNP*, 2018, pp. 132–142.

[11] M. Bagnulo, A. García-Martínez, and I. Van Beijnum, "The nat64/dns64 tool suite for ipv6 transition," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 177–183, 2012.

[12] J.-J. Lin, K.-C. Wang, S.-M. Cheng, and Y.-C. Liu, "On exploiting sdn to facilitate ipv4/ipv6 coexistence and transition," in *2017 IEEE conference on dependable and secure computing*. IEEE, 2017, pp. 473–474.

[13] Y. Cui, P. Wu, M. Xu, J. Wu, Y. L. Lee, A. Durand, and C. Metz, "4over6: network layer virtualization for ipv4-ipv6 coexistence," *IEEE Network*, vol. 26, no. 5, pp. 44–48, 2012.

[14] S. Li, Y. Shao, S. Ma, N. Xue, S. Li, D. Hu, and Z. Zhu, "Flexible traffic engineering: When openflow meets multi-protocol ip-forwarding," *IEEE Communications Letters*, vol. 18, no. 10, pp. 1699–1702, 2014.

[15] D. Gu, Y. Xue, D. Wang, and J. Li, "Ipv6 network virtualization architecture for autonomic management of ipv6 transition," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2017, pp. 625–631.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015, pp. 1346–1354.

[17] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *IEEE INFOCOM*, 2017, pp. 1–9.

[18] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "One step at a time: Optimizing sdn upgrades in isp networks," in *IEEE INFOCOM*, 2017, pp. 1–9.

[19] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *IEEE INFOCOM*, 2017, pp. 1–9.

[20] N. Herbaut, D. Negru, D. Dietrich, and P. Papadimitriou, "Service chain modeling and embedding for nfv-based content delivery," in *IEEE ICC*, 2017, pp. 1–7.

[21] B. Ren, D. Guo, G. Tang, X. Lin, and Y. Qin, "Optimal service function

tree embedding for nfv enabled multicast," in *IEEE ICDCS*, 2018, pp. 132–142.

[22] M. C. Luizelli, D. Raz, and Y. Sa'ar, "Optimizing nfv chain deployment through minimizing the cost of virtual switching," in *IEEE INFOCOM*, 2018, pp. 2150–2158.

[23] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *IEEE INFOCOM*, 2018, pp. 2132–2140.

[24] M. Blöcher, R. Khalili, L. Wang, and P. Eugster, "Letting off steam: Distributed runtime traffic scheduling for service function chaining," in *IEEE INFOCOM*, 2020.

[25] Y. Chen and J. Wu, "Nfv middlebox placement with balanced set-up cost and bandwidth consumption," in *ICPP*, 2018, pp. 1–10.

[26] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *IEEE INFOCOM*, 2019, pp. 523–531.

[27] N. Apollonio, R. I. Becker, I. Lari, F. Ricca, and B. Simeone, "Bicolored graph partitioning, or: gerrymandering at its worst," *Discrete Applied Mathematics*, vol. 157, no. 17, pp. 3601–3614, 2009.

[28] A. A. Bertossi, "Dominating sets for split and bipartite graphs," *Information Processing Letters*, vol. 19, no. 1, pp. 37–40, Jul. 1984.

[29] M. Chlebík and J. Chlebíková, "Approximation hardness of dominating set problems," in *Algorithms – ESA 2004*, 2004, pp. 192–203.

[30] R. Impagliazzo and R. Paturi, "Complexity of k-sat," in *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity*, 1999, pp. 237–240.

[31] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Springer International Publishing, 2015.

[32] P. Crescenzi, "A short guide to approximation preserving reductions," in *IEEE Conference on Computational Complexity*, 1997, pp. 262–273.

[33] I. Dinur and D. Steurer, "Analytical approach to parallel repetition," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, pp. 624–633.

[34] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015.

[35] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 399–404.

[36] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[37] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.