Virtual Network Embedding as Boolean Satisfiability

Pavel Surynek
Faculty of Information Technology
Czech Technical University
Prague, Czechia
pavel.surynek@fit.cvut.cz

Yi Zheng[1], Erik Kline[2], Sven Koenig[1], T. K. Satish Kumar[1,2]
[1]Department of Computer Science and [2]Information Sciences Institute
University of Southern California
Los Angeles, USA
yzheng63@usc.edu, kline@isi.edu, skoenig@usc.edu, tkskwork@gmail.com

***Abstract*—We address the Virtual Network Embedding (VNE) problem in which the task is to map a virtual network onto a given physical substrate network so that the CPU and bandwidth capacity constraints are met. Following the success of Boolean Satisfiability (SAT) methods in areas such as Multi-Agent Path Finding (MAPF), we propose in this paper a novel SAT-based approach for solving the VNE problem. As in MAPF, the various constraints that define the VNE problem are encoded into the SAT models incrementally and via lazy refinements so as to keep the models simple. We also propose various model relaxations and concomitant solution extraction post-processing procedures. Through experiments, we show that our SAT-based approach outperforms other state-of-the-art approaches on a number of VNE instances.**

***Index Terms*—Virtual Network Embedding, Boolean Satisfiability, Lazy Refinements**

I. Introduction

Network virtualization is an enabling technology that aims to overcome the Internet ossification problem, which refers to the resistance of the current Internet to architectural changes [1]. Through network virtualization, service providers can create multiple customized virtual networks to serve customers by leasing network resources from Infrastructure Providers (InPs) and without investing in changing the physical infrastructure. In addition, network virtualization facilitates increased security and manageability [2].

The physical infrastructure managed by InPs is often referred to as the Substrate Network (SN). The resourcefulness of an SN includes its CPU capacities, i.e., the compute capacities on its vertices, and its bandwidth capacities, i.e., the communication capacities on its edges. A virtual network requested by a customer is often referred to as a Virtual Network Request (VNR). In a VNR, each vertex is annotated with a CPU requirement, and each edge is annotated with a bandwidth requirement. The task is to allocate SN resources to satisfy the requirements of the VNR: Each VNR vertex must be mapped to an SN vertex (vertex/node mapping), and each VNR edge must be mapped to an SN path (edge/link mapping). The embedding must satisfy the CPU and bandwidth capacity constraints. It may also have to satisfy additional constraints that stem from geographical or other considerations. The problem of embedding a VNR onto an SN characterizes the main resource allocation task in network virtualization and is referred to as the Virtual Network Embedding (VNE) problem. The VNE problem and its many variants are NP-hard [3].

Recently, a Conflict-Based Search (CBS) algorithm has been developed to solve the VNE problem [4]. The solver, VNE-CBS, is inspired by the success of the CBS framework in the Multi-Agent Path Finding (MAPF) domain. In the MAPF problem [5], given an undirected graph and multiple agents with individual start and goal vertices, the task is to find a path for each agent from its start vertex to its goal vertex without conflicts (collisions) between any two paths. A conflict happens when two agents stay at the same vertex or traverse the same edge in opposite directions at the same time. Each action of an agent, either waiting at its current vertex or moving to a neighboring vertex, is assigned a cost. One of the common objectives is to minimize the sum of the costs incurred by the agents. Solving the MAPF problem optimally for this objective is NP-hard [6], [7]. There are many similarities between the MAPF problem and the VNE problem. These become apparent when the VNE problem is converted to a constrained path-coordination problem [8]. VNE-CBS is the first solver developed for the VNE problem that exploits these similarities and benefits from MAPF techniques.

In this paper, we follow the success of using Boolean Satisfiability (SAT) methods for solving the MAPF problem [9] and propose a novel SAT-based approach for solving the VNE problem. More concretely, we reduce a VNE instance to a series of SAT instances. In our SAT-based approach for solving the VNE problem, the various constraints that define the VNE problem are encoded into the SAT models incrementally and via lazy refinements so as to keep the models simple and the entire process efficient. Furthermore, we propose various model relaxations and enable simple post-processing procedures that extract solutions from other data structures returned by the SAT solver. Through experiments, we show that our SAT-based approach outperforms competing state-of-the-art approaches on a number of VNE instances.

II. Background

In this section, we provide the background literature on the VNE and MAPF problems.

A. Virtual Network Embedding

The VNE problem is essentially a constrained resource allocation problem that embeds (maps) a VNR onto an SN. An SN is an undirected graph $G^s = (V^s, E^s, A^s_V, A^s_E)$, where V^s is the set of SN vertices, E^s is the set of SN edges, A^s_V is a mapping from SN vertices to their attributes, and A^s_E is a mapping from SN edges to their attributes. The attributes of an SN vertex v^s include its CPU capacity $\textsc{cpu}(v^s)$ and its location $\textsc{loc}(v^s)$. The attribute of an SN edge e^s is its bandwidth capacity $\textsc{bw}(e^s)$. An SN path is a path in G^s. A VNR is an undirected graph $G^r = (V^r, E^r, C^r_V, C^r_E)$, where V^r is the set of VNR vertices, E^r is the set of VNR edges, C^r_V is a mapping from VNR vertices to their demands, and C^r_E is a mapping from VNR edges to their demands. The demands of a VNR vertex v^r include its CPU requirement $\textsc{cpu}(v^r)$ and the requirement of being mapped to an SN vertex that is within the maximum allowed distance $D(v^r)$ from a location attribute $\textsc{loc}(v^r)$. The demand of a VNR edge e^r is its bandwidth requirement $\textsc{bw}(e^r)$.

Given a VNR G^r and an SN G^s, the goal is to find a feasible VNE mapping, i.e., a mapping $\textsc{vne}(\cdot)$ of VNR vertices to SN vertices and VNR edges to SN paths. The mapping must satisfy a number of constraints: (a) each VNR vertex $v^r \in V^r$ is mapped to a unique and distinct SN vertex such that, for $v^r_i, v^r_j \in V^r$, $\textsc{vne}(v^r_i) = \textsc{vne}(v^r_j)$ if and only if $v^r_i = v^r_j$, (b) each VNR vertex $v^r \in V^r$ is mapped to an SN vertex such that $\textsc{cpu}(v^r) \leq \textsc{cpu}(\textsc{vne}(v^r))$ and $\textsc{geodist}(\textsc{loc}(v^r), \textsc{loc}(\textsc{vne}(v^r))) \leq D(v^r)$, where $\textsc{geodist}(\cdot,\cdot)$ is the geographical distance function between two locations, and (c) each VNR edge $(v^r_i, v^r_j) \in E^r$ is mapped to an SN path $\textsc{vne}((v^r_i, v^r_j))$ from $\textsc{vne}(v^r_i)$ to $\textsc{vne}(v^r_j)$ in G^s such that, for any SN edge $e^s \in E^s$, the sum of the bandwidth requirements of the VNR edges that utilize it does not exceed the bandwidth capacity of the SN edge, i.e., $\sum_{e^r \in E^r:\ e^s \in \textsc{vne}(e^r)} \textsc{bw}(e^r) \leq \textsc{bw}(e^s)$.

There are several popular metrics used to measure the quality of an embedding. The revenue refers to the sum of the virtual resources that are requested by the VNR and successfully meted out to it. That is, the revenue is given by $\sum_{v^r \in V^r} \textsc{cpu}(v^r) + \sum_{e^r \in E^r} \textsc{bw}(e^r)$. The cost refers to the sum of the SN resources that are allocated for embedding the VNR. That is, the cost is given by $\sum_{v^r \in V^r} \textsc{cpu}(v^r) + \sum_{e^r \in E^r} \sum_{e^s \in \textsc{vne}(e^r)} \textsc{bw}(e^r)$. Hence, the cost of embedding a VNR is at least its revenue.

The VNE problem has been formulated as a Mixed Integer Linear Programming (MILP) problem [8]. One approach relaxes the MILP formulation to a Linear Programming (LP) formulation and employs a deterministic or a randomized rounding technique to heuristically retrieve a solution from the fractional LP solution, resulting in two algorithms, D-ViNE and R-ViNE, respectively. These two algorithms are often used as the baseline for evaluating new VNE algorithms.

G-SP [10] and G-MCF [3] are two algorithms that first map VNR vertices to SN vertices greedily and then use shortest path or multi-commodity flow computations to map VNR edges to SN paths. Drawing inspiration from Google's Page Rank algorithm, RW-MaxMatch-SP sorts the SN and VNR vertices and maps them according to their ranks [11]. Then, it uses shortest path computations to map VNR edges to SN paths. The survey articles in [2] and [12] provide details of the VNE problem and its variants and classify many existing algorithms for solving them.

Inspired by the success of the CBS framework for solving the MAPF problem, a CBS algorithm VNE-CBS has been proposed for solving the VNE problem [4]. It converts the VNE problem to a constrained path-coordination problem as proposed in [8]. The resulting VNE-CBS solver conducts a two-level search. On the high level, VNE-CBS conducts a best-first search in conflict-resolution space: If the SN elements allocated to the VNR vertices and edges violate a constraint, it is resolved via branching. On the low level, VNE-CBS conducts a best-first search to repair the VNE mapping locally under the constraints imposed by the high-level search node. The success of VNE-CBS demonstrates that the VNE problem can benefit from importing and/or adapting the research conducted in the MAPF domain.

B. Multi-Agent Path Finding

The MAPF problem consists of k agents $\{a_1, a_2 \ldots a_k\}$ on a graph $G = (V, E)$, where each agent a_j has a start vertex $s^j \in V$ and a goal vertex $g^j \in V$. Time is discretized into time steps, and each agent can either move to a neighboring vertex or wait at its current vertex at each time step. Each action has a cost. A path of an agent is a sequence of move and wait actions that lead the agent from its start vertex to its goal vertex. A conflict arises when two agents are at the same vertex or traverse the same edge in opposite directions at the same time step. A solution of a MAPF instance is a set of paths without any conflicts. The commonly used objective is to minimize the sum of the costs incurred by all agents. The MAPF problem arises in many real-world applications, including video games [13], automated warehousing [14], and multi-drone delivery [15].

A successful approach for solving the MAPF problem optimally is via its reduction to a series of SAT instances and subsequent invocations of a SAT solver [9], [16]. The SAT problem [17] is the problem of determining whether a given Boolean formula in Conjunctive Normal Form (CNF) has an assignment under which the formula evaluates to $TRUE$. A formula in CNF is a conjunction of clauses, where a clause is a disjunction of literals and each literal is either a Boolean variable or its negation.

III. VNE as Path Coordination

In this section, we present the conversion of the VNE problem to a path-coordination problem and identify the similarities of the converted VNE problem to the MAPF problem.

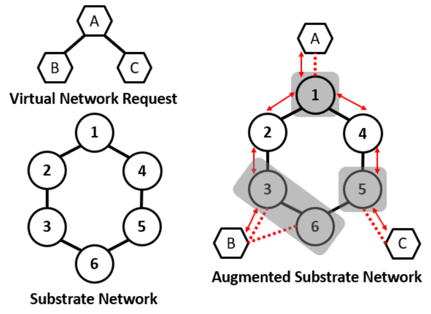

Fig. 1. An example of an augmented SN. The fictitious vertex B is connected via fictitious edges to the SN vertices 3 and 6 that satisfy its geographical constraint. The red arrows show the paths in the augmented SN for the VNR edges. Path $[A, 1, 2, 3, B]$ represents the vertex mappings of VNR vertices A and B and the edge mapping of VNR edge (A, B). A is mapped to the SN vertex 1, B is mapped to the SN vertex 3, and (A, B) is mapped to the SN path $[1, 2, 3]$.

The VNE problem can be reformulated as a constrained path-coordination problem on a newly created structure called the augmented SN [8]. To create the augmented SN $G^m = (V^m, E^m)$, where $V^m = V^s \cup V^f$ and $E^m = E^s \cup E^f$, we create a fictitious vertex $v^f \in V^f$ that represents each VNR vertex $v^r \in V^r$ and inherits all attributes of v^r, including $\text{CPU}(v^r)$, $\text{LOC}(v^r)$, and $D(v^r)$. Each v^f is connected via a fictitious edge $(v^f, v^s) \in E^f$ to each SN vertex v^s that satisfies the geographical constraint $\text{GEODIST}(\text{LOC}(v^f), \text{LOC}(v^s)) \leq D(v^f)$. The fictitious edges are given infinite bandwidth.

Consider a path $[v_i^f, v_1^s \dots v_2^s, v_j^f]$ in G^m, where v_i^f and v_j^f are the fictitious vertices created for v_i^r and v_j^r, respectively. The fictitious edges (v_i^f, v_1^s) and (v_2^s, v_j^f) correspond to the mapping of the VNR vertices v_i^r and v_j^r to the SN vertices v_1^s and v_2^s, respectively. The remaining SN edges correspond to the mapping of the VNR edge (v_i^r, v_j^r). Such a path is similar to an agent's path from its start vertex v_1^s to its goal vertex v_2^s in the MAPF domain. Figure 1 shows an example.

A feasible VNE mapping must satisfy a number of constraints. We capture the violations of these constraints as conflicts pertaining to the paths. A vertex conflict arises when two paths map the same VNR vertex to two different SN vertices or map two different VNR vertices to the same SN vertex. A CPU capacity conflict arises when a VNR vertex is mapped to an SN vertex that does not have sufficient CPU capacity to accommodate the CPU requirement of the VNR vertex. A bandwidth capacity conflict arises when one or more VNR edges utilize an SN edge that does not have sufficient bandwidth capacity to accommodate the bandwidth requirements of all the VNR edges.

IV. A SAT-Based Solver for the VNE Problem

Given a VNE instance on an SN $G^s = (V^s, E^s, A_V^s, A_E^s)$ with n vertices and a VNR $G^r = (V^r, E^r, C_V^r, C_E^r)$ with k vertices, a solution of our proposed SAT encoding should simultaneously establish the mapping of VNR vertices to SN vertices and the mapping of VNR edges to SN paths. It should also satisfy the geographical constraints, the CPU constraints, and the bandwidth constraints. Hence, we provide methods to satisfy each of the following: (a) the vertex and edge mapping constraints, (b) the geographical constraints, and (c) the CPU and bandwidth constraints.

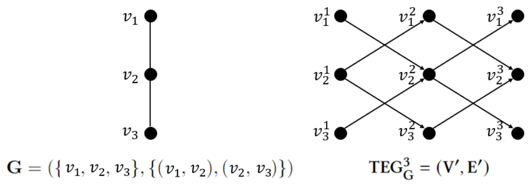

Fig. 2. An example of a TEG. TEG_G^3 represents paths in G of length at most 3. A path $[v_1, v_2, v_3]$ in G can be represented by a path $[v_1^1, v_2^2, v_3^3]$ in TEG_G^3.

A. Encoding Vertex and Edge Mapping Constraints

The mapping of VNR vertices to SN vertices is modeled via direct encoding (one-hot encoding). For each $v_i^r \in V^r$, $i \in \{1, 2 \dots k\}$, we introduce a set of Boolean variables $\mathcal{V}_i^j$, for $j \in \{1, 2 \dots n\}$, where $\mathcal{V}_i^j$ is $TRUE$ if and only if $v_i^r \in V^r$ is mapped to $v_j^s \in V^s$. To ensure that each VNR vertex is mapped to exactly one SN vertex, we introduce the following pseudo-Boolean constraints:

$$\sum_{j=1}^{n} \mathcal{V}_i^j = 1 \quad \forall i \in \{1, 2 \dots k\}. \tag{1}$$

In addition, we do not allow mapping two different VNR vertices to the same SN vertex. Therefore, we introduce the following constraints:

$$\sum_{i=1}^{k} \mathcal{V}_i^j \leq 1 \quad \forall j \in \{1, 2 \dots n\}. \tag{2}$$

To model the mapping of VNR edges to SN paths, we use a depth parameter $d \in \{2, 3 \dots n\}$. The depth specifies the maximum length of an SN path—measured by the number of vertices—to which a VNR edge can be mapped.

The encoding of the edge mapping is analogous to the encoding of paths in the SAT-based approach for solving the MAPF problem [9]. This approach uses the Time Expansion Graph (TEG). For a given undirected graph $G = (V, E)$ and depth d, TEG_G^d is a layered directed graph, where each layer corresponds to a copy of V and the interconnection between consecutive layers corresponds to a copy of E. Formally, we define the TEG as follows.

Definition 1: For $G = (V, E)$ with $V = \{v_1, v_2 \dots v_m\}$, the TEG of depth d is a directed graph $TEG_G^d = (V', E')$, where $V' = \{v_i^t \mid i \in \{1, 2 \dots m\} \wedge t \in \{1, 2 \dots d\}\}$ and $E' = \{\langle v_i^t, v_j^{t+1} \rangle \mid (v_i, v_j) \in E \wedge t \in \{1, 2 \dots d-1\}\}$. The set of vertices $\{v_i^t \mid i \in \{1, 2 \dots m\}\}$, for each $t \in \{1, 2 \dots d\}$, is called a layer.

Figure 2 shows an example of a TEG. In the VNE domain, we build TEGs of varying depth d for the SN. A TEG encodes the existence of a path between a pair of SN vertices to which the endpoint vertices of a VNR edge are mapped.

Let TEG^d_{SN} be the TEG of depth d constructed for the SN. For each VNR vertex $v^r_i \in V^r$, let $v^s_{\ell(i)} \in V^s$ be the SN vertex that it is mapped to via the constraints specified in Equations 1 and 2. For each VNR edge $(v^r_i, v^r_j) \in E^r$, the existence of a path from $v^s_{\ell(i)}$ to $v^s_{\ell(j)}$ in the SN must be encoded in the SAT model. If the length of any such path is restricted to be $\leq d$, its existence can be encoded via TEG^d_{SN}. Since TEG^d_{SN} is a directed layered graph, such a path should exist from $v^s_{\ell(i)}$ in the first layer to $v^s_{\ell(j)}$ in a subsequent layer. We use Boolean variables $\mathcal{X}(i,j)^t_l$ to encode such a path. Here, the superscript refers to the t-th layer of the TEG, and the subscript refers to the SN vertex v^t_l in this layer. $\mathcal{X}(i,j)^t_l$ is $TRUE$ if and only if v^t_l is included in the path.

To encode the required path, at most one SN vertex has to be chosen from each layer of the TEG. Hence, we have the following constraints for all $(v^r_i, v^r_j) \in E^r$:

$$\sum_{l=1}^{n} \mathcal{X}(i,j)^t_l \leq 1 \quad \forall t \in \{1, 2 \dots d\}. \tag{3}$$

To ensure that an SN path visits each SN vertex at most once, we have the following constraints for all $(v^r_i, v^r_j) \in E^r$:

$$\sum_{t=1}^{d} \mathcal{X}(i,j)^t_l \leq 1 \quad \forall l \in \{1, 2 \dots n\}. \tag{4}$$

The choice variables encoding the required path are related to the choice variables in Equations 1 and 2. Hence, we have the following constraints for all $(v^r_i, v^r_j) \in E^r$:

$$\mathcal{V}^l_i \rightarrow \mathcal{X}(i,j)^1_l \quad \forall l \in \{1, 2 \dots n\} \tag{5}$$

and

$$\mathcal{V}^l_j \rightarrow \bigvee_{t=2}^{d} \mathcal{X}(i,j)^t_l \quad \forall l \in \{1, 2 \dots n\}. \tag{6}$$

The forward propagation of the path in the TEG is encoded using the following constraints for all $(v^r_i, v^r_j) \in E^r$:

$$\mathcal{X}(i,j)^1_l \rightarrow \bigvee_{l' \mid (v^s_l, v^s_{l'}) \in E^s} \mathcal{X}(i,j)^2_{l'} \quad \forall l \in \{1, 2 \dots n\} \tag{7}$$

and

$$\mathcal{X}(i,j)^t_l \rightarrow \bigvee_{l' \mid (v^s_l, v^s_{l'}) \in E^s} \mathcal{X}(i,j)^{t+1}_{l'} \vee \mathcal{V}^l_j \quad \forall l \in \{1, 2 \dots n\}, \forall t \in \{2, 3 \dots d-1\}. \tag{8}$$

By symmetric reasoning in the reverse direction, an SN vertex in layer t can be reached from $v^s_{\ell(i)}$ in the first layer only via an SN vertex in layer $t-1$. Hence, we have the following constraints for all $(v^r_i, v^r_j) \in E^r$:

$$\mathcal{X}(i,j)^1_l \rightarrow \mathcal{V}^l_i \quad \forall l \in \{1, 2 \dots n\} \tag{9}$$

and

$$\mathcal{X}(i,j)^t_l \rightarrow \bigvee_{l' \mid (v^s_{l'}, v^s_l) \in E^s} \mathcal{X}(i,j)^{t-1}_{l'} \quad \forall l \in \{1, 2 \dots n\}, \forall t \in \{2, 3 \dots d\}. \tag{10}$$

Equations 1 to 10 can be converted to the clauses of a CNF formula $\mathcal{F}_{VNE}$. However, there are multiple ways to convert the pseudo-Boolean constraints of Equations 1, 2, 3, and 4 to CNF [18]–[20]. Here, we choose an adaptive encoding, which uses a pairwise encoding for a small number of variables but a sequential counter encoding for a large number of variables. We also note that $\mathcal{F}_{VNE}$ is derived from a fixed depth d of the TEG, which is iteratively increased in an outer loop.

B. Satisfying Geographical, CPU, and Bandwidth Constraints

Equations 1 to 10 encode the vertex mapping and edge mapping constraints of the VNE problem. They do not encode the geographical constraints, the CPU constraints, or the bandwidth constraints: These constraints are satisfied differently.

Adding geographical constraints to $\mathcal{F}_{VNE}$ is easy: We set $\mathcal{V}^j_i$ to $TRUE$ if and only if $\text{GEODIST}(\text{LOC}(v^r_i), \text{LOC}(v^s_j)) \leq D(v^r_i)$. However, the geographical constraints can also be controlled to exploit the power of incremental SAT-solving techniques: We first add tightened versions of the geographical constraints to reduce the search space and then incrementally relax them until we find a solution or the original geographical constraints are reinstated. Hence, in each iteration, we set $\mathcal{V}^j_i$ to $TRUE$ if and only if $\text{GEODIST}(\text{LOC}(v^r_i), \text{LOC}(v^s_j)) \leq g$. The parameter g is used as a threshold for all geographical constraints. Starting from a minimum value, it is incremented by a unit amount in each iteration, up to a maximum value g_{max}. However, it is not allowed to exceed $D(v^r_i)$ for the geographical constraint of any specific v^r_i. The iterations on the relaxation of the geographical constraints, i.e., the iterations that increase g, are nested within the iterations that increase the depth d of the TEG. The tightened geographical constraints can be incorporated in the SAT-solving procedure on $\mathcal{F}_{VNE}$ via the use of *assumptions* [21].

We satisfy the CPU and bandwidth capacity constraints of the VNE problem using *nogood recording*: a technique commonly used in lazy SAT encoding.[1] This technique is very effective in problem domains which have numerical constraints that express resource requirements and capacities. Eagerly encoding such numerical capacity constraints using Boolean variables is representationally unwieldy. In nogood recording, the idea is to only check the violations of these constraints instead of encoding them. If any such constraint is violated, the variables responsible for the violation are identified and recorded as a nogood. The nogood constraints are disjunctions that are simpler than the numerical capacity constraints. They are added back to the encoding of the problem for the next

[1]For the particular version of the VNE problem discussed in this paper, the CPU capacity constraints can be preprocessed to constraints on individual $\mathcal{V}^j_i$ variables since no more than one VNR vertex can be assigned to any SN vertex. However, our approach of recording nogoods serves the purpose of solving more general versions of the VNE problem as well.

iteration of SAT solving. This process is repeated until there are no more constraint violations or until unsatisfiability can be concluded. In the VNE domain, the CPU and bandwidth capacity constraints are not encoded explicitly in $\mathcal{F}_{VNE}$ but are enforced via nogood recording.[2]

A violation of a CPU capacity constraint arises if $\text{CPU}(v_i^r) > \text{CPU}(\text{VNE}(v_i^r))$. Suppose $v_j^s = \text{VNE}(v_i^r)$. The constraint violation can be eliminated by recording the nogood $\neg\mathcal{V}_i^j$. In the more general version of the VNE problem, a violated CPU capacity constraint may resemble $\text{CPU}(v_{i_1}^r) + \text{CPU}(v_{i_2}^r) + \text{CPU}(v_{i_3}^r) > \text{CPU}(v_j^s)$, where $v_{i_1}^r$, $v_{i_2}^r$, and $v_{i_3}^r$ are all mapped to v_j^s. In such a case, the violated constraint is resolved by recording the nogood $\neg\mathcal{V}_{i_1}^j \vee \neg\mathcal{V}_{i_2}^j \vee \neg\mathcal{V}_{i_3}^j$.

A violation of a bandwidth capacity constraint arises if one or more VNR edges utilize an SN edge that does not have the bandwidth capacity to accommodate them together. A nogood is recorded to resolve such a violation. For example, if three VNR edges $e_1^r = (v_{(1,a)}^r, v_{(1,b)}^r)$, $e_2^r = (v_{(2,a)}^r, v_{(2,b)}^r)$, and $e_3^r = (v_{(3,a)}^r, v_{(3,b)}^r)$ utilize the SN edge $e^s = (v_{j_1}^s, v_{j_2}^s)$ such that $\text{BW}(e_1^r) + \text{BW}(e_2^r) + \text{BW}(e_3^r) > \text{BW}(e^s)$, a nogood is recorded as follows. Suppose e_{inx}^r utilizes e^s between layers t_{inx} and $t_{inx}+1$ of $TEG_{G^s}^d$, for $inx \in \{1,2,3\}$, with the Boolean variables $\mathcal{X}((inx,a),(inx,b))_{j_1}^{t_{inx}}$ and $\mathcal{X}((inx,a),(inx,b))_{j_2}^{t_{inx}+1}$ set to $TRUE$. The recorded nogood is:

$$\begin{aligned}&\neg\mathcal{X}((1,a),(1,b))_{j_1}^{t_1} \vee \neg\mathcal{X}((1,a),(1,b))_{j_2}^{t_1+1}\\ \vee&\neg\mathcal{X}((2,a),(2,b))_{j_1}^{t_2} \vee \neg\mathcal{X}((2,a),(2,b))_{j_2}^{t_2+1}\\ \vee&\neg\mathcal{X}((3,a),(3,b))_{j_1}^{t_3} \vee \neg\mathcal{X}((3,a),(3,b))_{j_2}^{t_3+1}.\end{aligned}$$

Proposition 1: For a given VNE instance with G^s and G^r, $\mathcal{F}_{VNE}$ built on $TEG_{G^s}^d$, with all recorded nogoods for the CPU and bandwidth capacity constraints, is satisfiable if and only if there exists a feasible VNE mapping with embedded paths of length at most d.

Proof: '$\Leftarrow$': Consider a feasible VNE mapping $\text{VNE}(\cdot)$. We can set the variables in $\mathcal{F}_{VNE}$ according to $\text{VNE}(\cdot)$, which, by construction, satisfies Equations 1 to 10. Moreover, since $\text{VNE}(\cdot)$ satisfies all CPU and bandwidth capacity constraints, no recorded nogood prohibits it. Hence, $\mathcal{F}_{VNE}$ with all recorded nogoods is satisfiable. '$\Rightarrow$': We prove this by contradiction. Consider a satisfying assignment A of $\mathcal{F}_{VNE}$ with all recorded nogoods. A satisfies Equations 1 to 10 and, hence, correctly maps the VNR vertices to SN vertices via the $\mathcal{V}_i^j$ variables and the VNR edges to SN paths via the $\mathcal{X}(i,j)_l^t$ variables. If A does not encode a feasible VNE mapping, then it must violate a CPU or bandwidth capacity constraint. Hence, the corresponding nogood that is recorded is also violated. This contradicts our assumption that A satisfies $\mathcal{F}_{VNE}$ with all recorded nogoods.

Corollary 1: If $d = n$, then $\mathcal{F}_{VNE}$ with all recorded nogoods is satisfiable if and only if a feasible VNE mapping exists.

Proof: This follows from Proposition 1 and the fact that the maximum length of any path in G^s is n.

[2]This lazy encoding technique has also been shown to be successful in the MAPF domain [22].

Algorithm 1: VNE-SAT: an incremental SAT-solving procedure for the VNE problem.

```
Input: $G^s$, $G^r$, $d_{max}$
1  $g_{max} \leftarrow \max_{i=1}^{k} D(v_i^r)$
2  $d \leftarrow 2$
3  while $d \leq d_{max}$ do
4      $g \leftarrow 1$
5      while $g \leq g_{max}$ do
6          $VNE_{sol} \leftarrow$ VNE-SAT-bounded-geodist($G^s$, $G^r$, $d$, $g$)
7          if $VNE_{sol} \neq FAIL$ then
8              return $VNE_{sol}$
9          $g \leftarrow g + 1$
10     $d \leftarrow d + 1$
11 return $FAIL$
```

Algorithm 2: VNE-SAT-bounded-geodist: a SAT-solving procedure for the VNE problem with a fixed depth and bounded geographical distance.

```
Input: $G^s$, $G^r$, $d$, $g$
1  $\mathcal{F}_{VNE} \leftarrow$ build-VNE-SAT-model($G^s$, $G^r$, $d$, $g$)
2  while $TRUE$ do
3      $sol \leftarrow$ solve-SAT-instance($\mathcal{F}_{VNE}$)
4      if $sol \neq FAIL$ then
5          $VNE_{asgn} \leftarrow$ extract-VNE-assignment($sol$)
6          $U^r \leftarrow \{v_i^r \in V^r \mid \text{CPU}(v_i^r) > \text{CPU}(\text{VNE}(v_i^r))\}$
7          $F^s \leftarrow \{e^s \in E^s \mid \sum_{e^r \in E^r:\ e^s \in \text{VNE}(e^r)} \text{BW}(e^r) > \text{BW}(e^s)\}$
8          if $U^r = \emptyset \wedge F^s = \emptyset$ then
9              return $VNE_{asgn}$
10         else
11             for $v_i^r \in U^r$ do
12                 $\mathcal{F}_{VNE} \leftarrow \mathcal{F}_{VNE} \wedge$ generate-CPU-nogood($v_i^r$, $VNE_{asgn}$)
13             for $e^s \in F^s$ do
14                 $\mathcal{F}_{VNE} \leftarrow \mathcal{F}_{VNE} \wedge$ generate-BW-nogood($e^s$, $VNE_{asgn}$)
15     else
16         return $FAIL$
```

C. Our Proposed Algorithm: VNE-SAT

Algorithm 1 shows the pseudocode of our proposed SAT-based algorithm, which we call VNE-SAT. VNE-SAT uses the same practitioners' intuitions and working principles as SAT solvers used in automated planning. It is more promising to start with small SAT instances since the complexity of solving SAT instances typically increases exponentially with the number of variables. Hence, the idea in VNE-SAT is to create SAT encodings with iteratively increasing values of d in the outer loop and iteratively increasing values of g in the inner loop. Here, $d \in \{2, 3 \dots n\}$ is the depth parameter in TEG_{SN}^d and $g \in \{1, 2 \dots g_{max}\}$ is the threshold employed for all geographical constraints.

On Lines 1–2, Algorithm 1 initializes g_{max} and d. It increments d in the outer loop on Lines 3–10. It increments g in the inner loop on Lines 5–9. On Line 6, it calls the function VNE-SAT-bounded-geodist to check the feasibility of the problem for a given depth d and all geographical distances bounded by g. If this function finds a solution, Algorithm 1 returns it on Line 8. Else, the loops continue until termination; and, on Line 11, the algorithm returns failure to find a solution.

Algorithm 2 shows the pseudocode of the function VNE-SAT-bounded-geodist. It uses fixed values of d and g. On Line 1, it builds the SAT encoding $\mathcal{F}_{VNE}$ of the VNE problem for the specified values of d and g using Equations 1 to 10. On Lines 2–16, it tries to satisfy the CPU and bandwidth capacity constraints using nogood recording. On Line 3, it tries to solve $\mathcal{F}_{VNE}$ using an off-the-shelf complete SAT solver. If $\mathcal{F}_{VNE}$ is reported to not admit a solution, the algorithm immediately reports a failure on Line 16. Else, if $\mathcal{F}_{VNE}$ admits a solution, the algorithm first extracts the VNE assignment corresponding to this solution on Line 5. Then, on Lines 6–7, the algorithm checks for any violated CPU or bandwidth capacity constraints and stores them in U^r and F^s, respectively. If both U^r and F^s are empty, then the extracted VNE assignment is a valid solution, which the algorithm returns on Line 9. Else, for each violated CPU and bandwidth capacity constraint, the algorithm adds a nogood to $\mathcal{F}_{VNE}$ on Lines 12 and 14, respectively, as described before.

The following corollary establishes the completeness of VNE-SAT in Algorithm 1.

Corollary 2: VNE-SAT is complete with respect to the specified maximum length d_{max} of the SN paths implementing the VNR edges. For $d_{max} = n$, the algorithm is complete for the VNE problem.

D. Model Relaxations

While the CPU and bandwidth capacity constraints are handled via nogood recording, the other requirements of the VNE problem are encoded in Equations 1 to 10. However, Equations 3 and 4 introduce a lot of clauses in $\mathcal{F}_{VNE}$ since they are pseudo-Boolean constraints that involve summations. Their expansion to SAT clauses can be impeding and unwieldy. Hence, we consider methods that circumvent these equations.

Removing Equation 3 results in the possibility of a VNR edge not being implemented as a proper SN path. In particular, multiple $\mathcal{X}(i,j)^t_l$ variables can be $TRUE$ for the same i, j, and t. As a result, every $\mathcal{X}(i,j)^t_l$ variable that is $TRUE$ may have multiple predecessors at layer $t-1$ and multiple successors at layer $t+1$ that are also $TRUE$. However, the other equations still enforce a structure on the combination of variables that can be $TRUE$. In particular, Equation 4 still ensures that every SN vertex is chosen at most once. Hence, we refer to this relaxation as the *tree relaxation*.

It is also possible to remove Equation 4. While Equation 4 enforces the absence of repeating vertices from the SN paths that implement the VNR edges, its removal allows for cycles in the output structure. Hence, removing Equations 3 and 4 is referred to as the *graph relaxation*.

While the tree and the graph relaxations do not necessarily yield the embeddings of the VNR edges, they yield output structures that are still useful.

Proposition 2: The original SAT model, its tree relaxation, and its graph relaxation, are all equisatisfiable, i.e., a solution exists for one model if and only if a solution exists for the other models.

Proof: We first prove that if the graph relaxation admits a solution then so does the original model. Although the graph relaxation removes Equations 3 and 4, the other equations in it enforce that the endpoints of each VNR edge are still reachable from one another in the SN. This means that there exists an SN path between the endpoints. Hence, a solution exists for the original model. All other conditions required for equisatisfiability follow from the foregoing proof and the following simple argument: If a solution exists for a certain model, then a solution exists for any relaxation of that model.

The tree and the graph relaxations significantly reduce the size of the SAT instances. However, both relaxations necessitate a post-processing step in solution extraction for the original VNE problem: A search procedure is required to be carried out on the generated output structures to find a valid SN path that implements each VNR edge.

E. An Alternative SAT Model not Based on the TEG

We note that the TEG has a layer for each time step, in which there is a vertex corresponding to each SN vertex. Such a representation is justified in the MAPF domain since an agent can visit a single vertex multiple times. However, in the VNE domain, an SN path does not have multiple occurrences of the same SN vertex. Hence, it is possible to design an alternative SAT model which does not use the TEG.

In the new SAT model, sets of Boolean variables are introduced in correspondence to the SN G^s. For each VNR edge $(v^r_i, v^r_j) \in E^r$, a set of Boolean variables encodes an SN path for it. In this set, the variable $\mathcal{X}(i,j)_l$ is $TRUE$ if and only if the SN vertex $v^s_l \in V^s$ is chosen to be on the SN path. Moreover, the Boolean variables $\mathcal{E}(i,j)_{(l_1,l_2)}$, for all $(v^r_i, v^r_j) \in E^r$ and $(v^s_{l_1}, v^s_{l_2}) \in E^s$, are introduced so that $\mathcal{E}(i,j)_{(l_1,l_2)}$ is $TRUE$ if and only if the SN edge $(v^s_{l_1}, v^s_{l_2})$ is chosen to be on the SN path that implements (v^r_i, v^r_j). We use $\mathcal{F}_{FLAT}$ to denote the resulting SAT model. Although $\mathcal{F}_{FLAT}$ does not use time expansion, it may introduce a large number of variables corresponding to the edges of the SN. $\mathcal{F}_{FLAT}$ introduces the following constraints.

If an SN edge $(v^s_{l_1}, v^s_{l_2})$ is selected, then both its endpoints should be selected. Hence, we have:

$$\mathcal{E}(i,j)_{(l_1,l_2)} \rightarrow \mathcal{X}(i,j)_{l_1} \wedge \mathcal{X}(i,j)_{l_2} \tag{11}$$

If an SN vertex v^s_l is selected as an intermediate vertex on the SN path, then exactly one incoming edge to it and exactly one outgoing edge from it should be selected. Hence, we have:

$$\mathcal{X}(i,j)_{l_1} \rightarrow \bigvee_{l_2 \mid (v^s_{l_1}, v^s_{l_2}) \in E^s} \mathcal{E}(i,j)_{(l_1,l_2)} \vee \mathcal{V}^{l_1}_j \tag{12}$$

$$\sum_{l_2 \mid (v^s_{l_1}, v^s_{l_2}) \in E^s} \mathcal{E}(i,j)_{(l_1,l_2)} + \mathcal{V}^{l_1}_j \leq 1 \tag{13}$$

$$\mathcal{X}(i,j)_{l_2} \rightarrow \bigvee_{l_1 \mid (v^s_{l_1}, v^s_{l_2}) \in E^s} \mathcal{E}(i,j)_{(l_1,l_2)} \vee \mathcal{V}^{l_2}_i \tag{14}$$

$$\sum_{l_1 \mid (v^s_{l_1}, v^s_{l_2}) \in E^s} \mathcal{E}(i,j)_{(l_1,l_2)} + \mathcal{V}^{l_2}_i \leq 1 \tag{15}$$

V. Experiments

In this section, we present an empirical evaluation of VNE-SAT against popular VNE algorithms that are standardly used as baseline procedures to evaluate new VNE algorithms. These include VNE-CBS, D-ViNE, R-ViNE, G-SP, and RW-MaxMatch-SP. We use VNE-CBS with the suboptimality factor $w = 2.0$. In our experiments, we denote the vanilla version of Algorithm 1 by VNE-SAT. We denote the tree and the graph relaxations of it by VNE-SAT-TREE and VNE-SAT-GRAPH, respectively. When these procedures are used with a fixed depth $d = 10$, we denote them by VNE-SAT-FD, VNE-SAT-FD-TREE, and VNE-SAT-FD-GRAPH, respectively. When Algorithm 1 uses $\mathcal{F}_{FLAT}$ instead of $\mathcal{F}_{VNE}$, we denote it by FLAT.

We implemented our algorithms in C++ and used Glucose 3.0 [23] as the underlying SAT solver. Glucose is used to solve SAT instances incrementally [21]. After solving a SAT instance, Glucose preserves the state of the solver, including the learned clauses, to facilitate incrementally solving a new SAT instance that may differ only marginally from the previously solved SAT instance. This is useful for Algorithm 1.

We used a standard methodology from the VNE literature to generate VNE instances via Waxman graphs. Waxman graphs [24] are frequently chosen in simulations as topologies that resemble communication networks. We generated SN topologies as random Waxman graphs in a 50×50 grid space with the parameters $\alpha = 0.5$ and $\beta = 0.2$. We generated 5 SNs, each with 100 vertices that resulted in 511, 530, 556, 576, and 594 edges, respectively. The CPU and bandwidth capacities of the SN vertices and the SN edges were set to real numbers drawn uniformly at random from the interval $[50, 100]$. We also generated VNR topologies in the same manner. We generated $1,000$ VNRs in each of four categories: with 10, 20, 30, and 40 VNR vertices. For each VNR vertex v^r, we set $D(v^r)$ to be 15. The CPU and bandwidth requirements were drawn uniformly at random from the intervals $[0, 20]$ and $[0, 50]$, respectively. Our VNE instances are significantly larger than the ones commonly used in previous work since our proposed algorithm is more scalable.

We ran all experiments on an AWS machine with 8 CPUs and 16 GB RAM. We set a timeout of 60 seconds for each VNE instance.

A. *Offline Experiments*

In this subsection, we present experimental results from an offline setting. In this setting, we used $5,000$ VNE instances from all possible combinations of the generated 5 SNs and $1,000$ VNRs. For each competing algorithm, we report the success rate as well as the average cost and the average runtime. The success rate is the percentage of successfully solved instances. The average cost measures the SN resources allocated for the VNR embedding, averaged over all successfully solved instances. The average runtime is also computed only over all successfully solved instances.

Figure 3 shows the results of our experiments. On the top row, it presents comparative results for the various versions of our proposed VNE-SAT. On the bottom row, it presents comparative results for the top-performing versions of VNE-SAT against the standardly used baseline VNE algorithms.

On the top row, we observe that the VNE-SAT algorithms have better success rates compared to the VNE-SAT-FD algorithms. FLAT has the worst performance. The same trends are also true on the metrics of average cost and average runtime. These observations indicate that the incremental SAT-solving methodology is very beneficial. VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH achieve nearly 100% success rates even for embedding VNRs with 40 vertices. Also, they do so while maintaining a low average cost and a short runtime.

On the bottom row, we observe that the VNE-SAT algorithms also have better performance compared to other standard VNE algorithms. For VNRs with 40 vertices, VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH achieve 99.74%, 99.80%, and 99.82% success rates, respectively, marginally outperforming RW-MaxMatch-SP, which has a success rate of 99.4%, and significantly outperforming all other algorithms. D-ViNE and R-ViNE fail to solve many instances with 30 VNR vertices, and VNE-CBS fails to solve many instances with 40 VNR vertices. On the metric of average cost, VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH are close to the top-performing VNE-CBS. Their average costs are much lower than those of G-SP and RW-MaxMatch-SP. On the metric of average runtime, VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH yield 2.53, 2.56, and 2.38 seconds, respectively, for VNRs with 40 vertices. These numbers are marginally larger than those of G-SP and RW-MaxMatch-SP, since these latter algorithms are greedy. The overall superior performance of our proposed VNE-SAT algorithms over competing VNE algorithms can be attributed to the efficient algorithmic techniques packaged in modern SAT solvers.

B. *Online Experiments*

In this subsection, we present the experimental results from an online setting. In this setting, we embed a series of VNRs that arrive at different time steps, with the assumption that each successfully embedded VNR holds the SN resources allocated to it until it departs at the end of its lifetime. We generated the arrival times of the VNRs according to a Poisson process at an average arrival rate of 4 VNRs per 100 time steps. The lifetime of each VNR was drawn from an exponential distribution with an average lifetime of 100 time steps. We do not allow for the reconfiguration of any previously embedded VNRs that still hold the SN resources. If an incoming VNR cannot be embedded within 60 seconds, the VNE algorithm rejects it and tries the next VNR. We used 5 runs, each corresponding to one of the 5 SNs. All algorithms were run on the same $1,000$ VNRs for each category of the number of VNR vertices.

For each competing algorithm, we report the average acceptance ratio, the average total revenue, and the average cost/revenue ratio. The acceptance ratio is the percentage of VNRs that are successfully embedded. The average acceptance ratio is the acceptance ratio averaged over the 5 runs. The total revenue measures the sum of the revenues from all

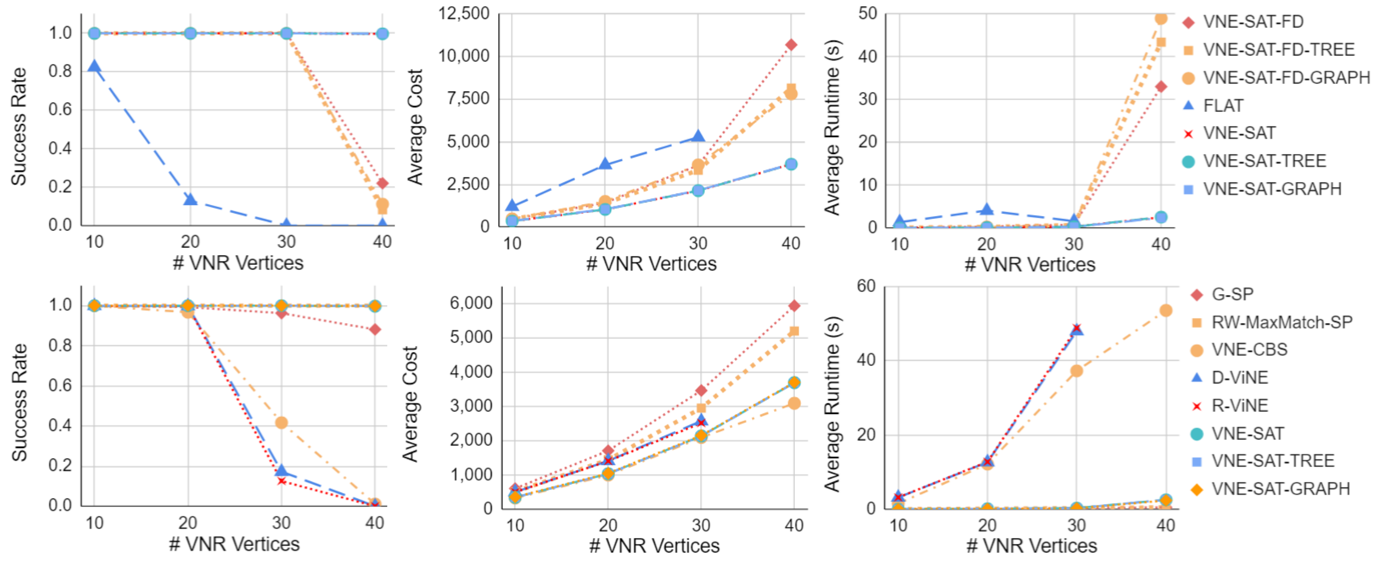

Fig. 3. Offline Experiments: the success rate, the average cost, and the average runtime (seconds) across different numbers of VNR vertices.

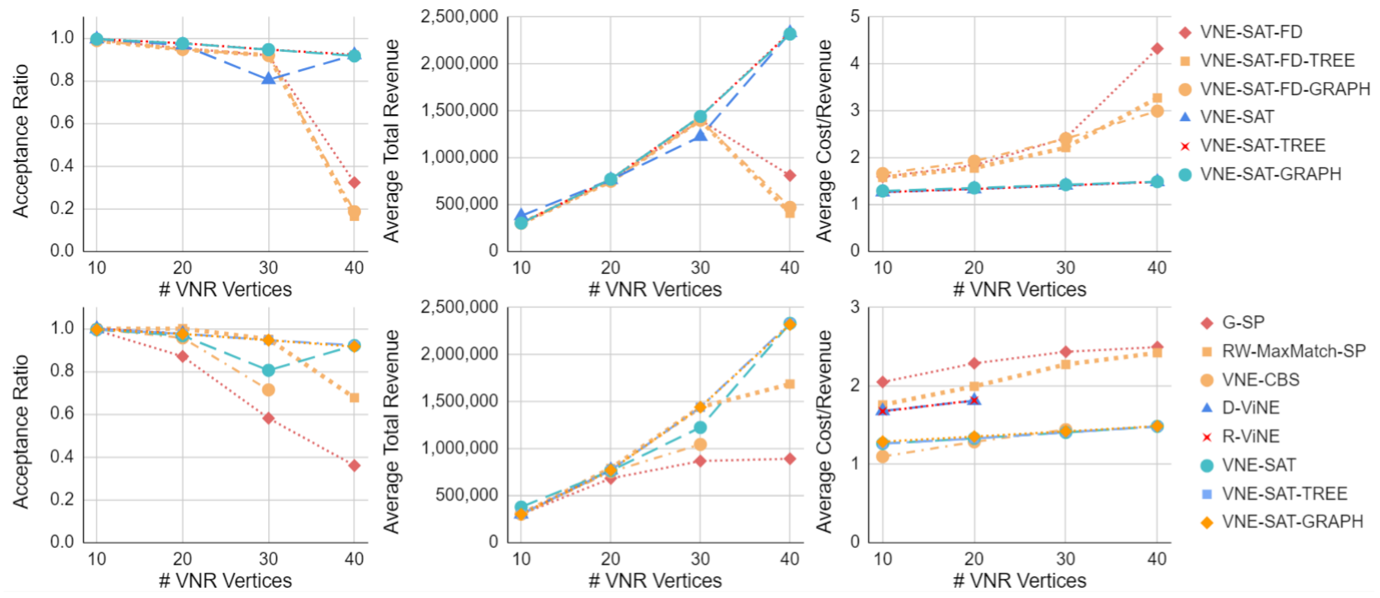

Fig. 4. Online Experiments: the acceptance ratio, the average total revenue, and the average cost/revenue across different numbers of VNR vertices.

successfully embedded VNRs. The average total revenue is the total revenue averaged over the 5 runs. The cost/revenue is the cost divided by the revenue for each accepted VNR. The average cost/revenue is the cost/revenue averaged over all VNRs accepted across the 5 runs.

Figure 4 shows the results of our experiments. On the top row, it presents comparative results for the various versions of our proposed VNE-SAT. On the bottom row, it presents comparative results for the top-performing versions of VNE-SAT against the standardly used baseline VNE algorithms. We excluded FLAT from the online experiments since its performance is very poor even in the offline setting.

On the top row, we observe that the VNE-SAT algorithms significantly outperform the VNE-SAT-FD algorithms on all metrics. These observations demonstrate the benefits of incremental SAT-solving methodology even in the online setting.

On the bottom row, we observe that the VNE-SAT algorithms also have better performance compared to other standard VNE algorithms. On the metric of average acceptance ratio, VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH achieve the highest performances. For VNRs with 40 vertices, they achieve 92.14%, 92.18%, and 91.60% average acceptance ratios, respectively. However, D-ViNE, R-ViNE, and VNE-CBS do not scale as the number of vertices increases. On the metrics of average total revenue and average cost/revenue ratio, VNE-SAT, VNE-SAT-TREE, and VNE-SAT-GRAPH once again achieve the highest performances. These observations are indicative of our proposed VNE-SAT algorithms being able to use the SN resources much more effectively than other VNE algorithms.

VI. Conclusions

The VNE problem arises as a cornerstone optimization problem in the efficient and effective management of resources on a communication network. In this paper, we presented a SAT-based approach for solving the VNE problem. Our approach used several ideas. First, we found it beneficial to build the SAT model incrementally and harness the power of incremental SAT-solving techniques. Second, we used lazy refinements and nogood recording to handle CPU and bandwidth capacity constraints efficiently and effectively. Third, we proposed relaxation techniques with concomitant solution extraction post-processing procedures to improve the efficiency of the SAT encoding. We encapsulated all these techniques in a solver called VNE-SAT and its variants. Through experiments on a number of VNE instances, we showed that VNE-SAT and its variants comfortably outperform other existing state-of-the-art VNE algorithms in both the offline and the online settings.

VII. Acknowledgements

This research was supported by DARPA under grant number HR001120C0157, by NSF under grant numbers 2112533, 2434916, 2346058, 2321786, 2121028, and 1935712, by gifts from Amazon Robotics, and by the Czech Science Foundation under project number 22-31346S.

References

[1] M. Chowdhury and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges," *IEEE Communications Magazine*, 2009.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Communications Surveys and Tutorials*, 2013.

[3] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," *Computer Communication Review*, 2008.

[4] Y. Zheng, S. Ravi, E. Kline, S. Koenig, and T. K. S. Kumar, "Conflict-Based Search for the Virtual Network Embedding Problem," in *International Conference on Automated Planning and Scheduling*, 2022.

[5] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," in *Symposium on Combinatorial Search*, 2019.

[6] J. Yu and S. M. LaValle, "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs," in *AAAI Conference on Artificial Intelligence*, 2013.

[7] H. Ma, C. A. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig, "Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem," in *AAAI Conference on Artificial Intelligence*, 2016.

[8] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *Joint Conference of the IEEE Computer and Communications Societies*, 2009.

[9] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective," in *European Conference on Artificial Intelligence*, 2016.

[10] Y. Zhu and M. H. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Joint Conference of the IEEE Computer and Communications Societies*, 2006.

[11] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual Network Embedding through Topology-Aware Node Ranking," *Computer Communication Review*, 2011.

[12] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A Survey of Embedding Algorithm for Virtual Network Embedding," *China Communications*, 2019.

[13] D. Silver, "Cooperative Pathfinding," in *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2005.

[14] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Magazine*, 2008.

[15] S. Choudhury, K. Solovey, M. J. Kochenderfer, and M. Pavone, "Efficient Large-Scale Multi-Drone Delivery Using Transit Networks," in *IEEE International Conference on Robotics and Automation*, 2020.

[16] P. Surynek, R. Stern, E. Boyarski, and A. Felner, "Migrating Techniques from Search-Based Multi-Agent Path Finding Solvers to SAT-Based Approach," *Journal of Artificial Intelligence Research*, 2022.

[17] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Structures in Constructive Mathematics and Mathematical Logic*, 1968.

[18] O. Bailleux and Y. Boufkhad, "Efficient CNF Encoding of Boolean Cardinality Constraints," in *Principles and Practice of Constraint Programming*, 2003.

[19] J. P. M. Silva and I. Lynce, "Towards Robust CNF Encodings of Cardinality Constraints," in *Principles and Practice of Constraint Programming*, 2007.

[20] V. Nguyen and S. T. Mai, "A New Method to Encode the At-Most-One Constraint into SAT," in *International Symposium on Information and Communication Technology*, 2015.

[21] G. Audemard, J. Lagniez, and L. Simon, "Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction," in *International Conference on Theory and Applications of Satisfiability Testing*, 2013.

[22] P. Surynek, "Unifying Search-Based and Compilation-Based Approaches to Multi-Agent Path Finding through Satisfiability Modulo Theories," in *International Joint Conference on Artificial Intelligence*, 2019.

[23] G. Audemard and L. Simon, "On the Glucose SAT Solver," *International Journal on Artificial Intelligence Tools*, 2018.

[24] B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications*, 1988.