Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Conflict-Based Search for the Virtual Network Embedding Problem

Yi Zheng, Srivatsan Ravi, Erik Kline, Sven Koenig, T. K. Satish Kumar

University of Southern California
yzheng63@usc.edu, sravi@isi.edu, kline@isi.edu, skoenig@usc.edu, tkskwork@gmail.com

Abstract

In emerging network virtualization architectures, service providers will be able to create many heterogeneous virtual networks and offer customized end-to-end services by leasing shared resources from infrastructure providers. The Virtual Network Embedding (VNE) problem is central to such technology. It involves the proper allocation of CPU and bandwidth resources available in a physical substrate network to meet the demands of multiple virtual networks. Combinatorially, the VNE problem is a problem in resource management that is NP-hard to solve. In this paper, we present a novel version of the Conflict-Based Search (CBS) algorithm for solving the VNE problem. Our approach, called VNE-CBS, is inspired by the success of the CBS framework in the Multi-Agent Path Finding domain. We successfully address the unique challenges in applying the CBS framework to the VNE problem, and, in doing so, we pave the way for overcoming a crucial issue in Internet ossification via heuristic search methods. On the theoretical front, we show that, unlike many existing algorithms, our algorithm is complete and optimal. On the experimental front, we show that our algorithm significantly outperforms other state-of-the-art methods on various benchmark instances for both the offline and online versions of the VNE problem.

Introduction

The Internet ossification problem refers to the resistance of the current Internet to architectural changes. Network virtualization is an enabling technology that addresses this problem (Chowdhury and Boutaba 2009). It does so by modularizing Internet service provision to (a) infrastructure provision and (b) end-to-end service provision. In other words, Infrastructure Providers (InPs) are only concerned with managing the physical infrastructure, and Service Providers (SPs) are only concerned with gleaning resources from multiple InPs to create Virtual Networks (VNs) and provide end-to-end services.

Network virtualization facilitates flexibility and diversity since heterogeneous network architectures can be deployed despite ossifications in the physical infrastructure. In addition, network virtualization facilitates increased security and manageability. In a network virtualization environment, multiple SPs can create heterogeneous VNs and offer customized end-to-end services to users by leasing shared resources from one or more InPs without significant investment in the physical infrastructure (Feamster, Gao, and Rexford 2007). The physical infrastructure managed by InPs is often referred to as the Substrate Network (SN). Network virtualization can also be used to evaluate multiple new network protocols simultaneously on a shared experimental facility (Anderson et al. 2005).

Although network virtualization has many benefits, it hinges on an efficient and effective coordination of SPs and InPs. This translates to the need for an efficient and effective management of the SN capacities. The capacities of an SN include its CPU capacity, i.e., the compute power on its vertices, and its bandwidth capacity, i.e., the communication capacities on its edges.

A request for SN capacities from an SP appears in the form of a VN and is referred to as a Virtual Network Request (VNR). In a VNR, vertices are annotated with their CPU requirements, and edges are annotated with their bandwidth requirements. Each VNR vertex is required to be mapped to an SN vertex, and each VNR edge is required to be mapped to an SN path. These mappings are required to satisfy various constraints, such as CPU, bandwidth, and geographical constraints.

The above problem of embedding VNRs in an SN is referred to as the Virtual Network Embedding (VNE) problem. It is a combinatorial problem that models the proper management and allocation of shared resources on a network. It is foundational to Edge Computing and 5G Networks. In these contexts, it feeds into Networking Slicing, a mechanism used to provision resources across multiple infrastructures via a VN that meets certain quality-of-service or service-level agreement requirements. The VNE problem and its many variants are NP-hard (Yu et al. 2008).

In this paper, we present a novel Conflict-Based Search (CBS) algorithm for efficiently solving the VNE problem. Our approach is inspired by the success of the CBS framework in the Multi-Agent Path Finding (MAPF) domain. The MAPF problem (Stern et al. 2019) arises in many real-world applications, including video games (Silver 2005), automated warehousing (Wurman, D'Andrea, and Mountz 2008), multi-drone delivery (Choudhury et al. 2020), and aircraft-towing vehicles (Morris et al. 2016).

In the MAPF problem, each agent is required to move from a start vertex to a goal vertex on a directed or undirected graph while avoiding collisions (also called conflicts) with other agents. 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, such as moving to a neighboring vertex or staying at its current vertex, has a cost. One of the common objectives for the MAPF problem is minimizing the sum of the travel costs of the agents. Solving the MAPF problem optimally for this objective is NP-hard (Yu and LaValle 2013; Ma et al. 2016).

CBS is a two-level search algorithm for solving the MAPF problem optimally (Sharon et al. 2015). It starts by planning a path for each agent independently. If a conflict arises between two agents, CBS resolves it by branching on two possibilities, with each possibility spatiotemporally constraining the motion of one of the two conflicting agents. Thus, the high-level search builds a conflict resolution tree, referred to as the Constraint Tree (CT). Each CT node accumulates a set of spatiotemporal constraints. The low-level search performs single-agent pathfinding to compute optimal individual paths under the spatiotemporal constraints imposed by the high level. The CBS search framework is the basis of many successful MAPF solvers (Sharon et al. 2015; Barer et al. 2014).

Recognizing the combinatorial similarities and the subtle differences between the VNE problem and the MAPF problem, we develop a CBS-based solver for the VNE problem, called VNE-CBS. On one hand, VNE-CBS exploits the similarities between the VNE problem and the MAPF problem. On the other hand, it also successfully addresses the subtle differences and unique challenges in applying the CBS framework to the VNE problem. We show that, unlike many existing algorithms, VNE-CBS is complete and optimal. We also develop a bounded-suboptimal version of VNE-CBS that trades off optimality for increased efficiency. Through a wide range of experiments, we show that our CBS-based algorithms significantly outperform other state-of-the-art methods on various benchmark instances, for both the offline and online versions of the VNE problem.

Overall, the significance of our methodology is that it paves the way for overcoming Internet ossification via heuristic search methods.

Background

VNE-CBS relies on three major conceptual components: VNE, MAPF, and CBS. In this section, we describe the background literature relevant to each of them.

VNE

The VNE problem is essentially the problem of embedding a VNR in a real physical SN such that the VNR vertices are realized as SN vertices with satisfactory compute power and the VNR edges are realized as SN paths with satisfactory bandwidth capacity. Naturally, these mappings are required to satisfy various constraints, such as: (a) For each SN vertex, the sum of the CPU requirements of all VNR vertices mapped to it should be no greater than its available CPU capacity. (b) For each SN edge, the sum of the bandwidth requirements of all VNR edges that utilize it should be no greater than its available bandwidth capacity. Other requirements may also exist, such as: (c) Each VNR vertex may be geographically constrained, having to be mapped to an SN vertex within a specified subset of the SN vertices.

Overall, the VNE problem is a constrained resource allocation problem. It can be reduced to the multiway separator problem, making it NP-hard (Andersen 2002). There are many variants of the above "core" VNE problem that are also NP-hard (Yu et al. 2008). In particular, there are several optimization variants of the VNE problem that are defined for different metrics such as revenue, cost, ratio of cost to revenue, acceptance ratio, and the running time of the VNE algorithm. The revenue refers to the sum of CPU and bandwidth capacities requested by a VNR, while the cost refers to the CPU capacities and the bandwidth capacities spent by the SN to map a VNR. The acceptance ratio refers to the fraction of VNRs, from a sequence of VNRs, that can be embedded in the SN.

In the offline version of the VNE problem, a collection of VNRs are required to be embedded in the SN. The objective is to maximize the acceptance ratio while heeding to other metrics. In real-world scenarios, VNRs arrive in sequence at different times and stay in the network for an arbitrary duration of time. This is the online version of the VNE problem. In this case, reconfiguration may be necessary, i.e., it may be necessary to backtrack on the embedding assignments made for past VNRs to be able to accommodate new VNRs when they arrive.

Several other variants of the VNE problem, be it offline or online, can be defined using various other considerations (Fischer et al. 2013). For example, delays can be considered in addition to bandwidth requirements. The path-splitting version of the VNE problem allows for a VNR edge to be realized using multiple SN paths—instead of a single SN path—as far as the total bandwidth requirement is satisfied. In the "redundant" version of the VNE problem, an embedding requires additional resources for VNRs to accommodate for potential substrate resource failures.

Many algorithms have been proposed to solve the core VNE problem and its variants. Chowdhury, Rahman, and Boutaba (2009) formulate the VNE problem with geographical constraints as a Mixed Integer Linear Programming (MILP) problem. They further relax the MILP problem to a Linear Programming (LP) problem and introduce two new algorithms. These are a deterministic rounding algorithm, called D-ViNE, and a randomized rounding algorithm, called R-ViNE, that are used to heuristically retrieve a solution from the fractional solution of the relaxed LP problem. D-ViNE and R-ViNE are used as standard baseline algorithms for evaluating new VNE algorithms.

Some VNE algorithms use node ranking. For example, drawing inspiration from Google's Page Rank algorithm, Cheng et al. (2011) sort the SN and VNR vertices and map them according to their ranks. Then, they use rules to map the VNR edges onto SN paths. Fischer et al. (2013) and Cao et al. (2019) provide a detailed survey of the VNE problem, its variants, and a classification of existing approaches

for solving the VNE problem.

There are also several simulation tools available for evaluating VNE algorithms. ViNEYard (Chowdhury, Rahman, and Boutaba 2012) is a popular VNE simulation tool that uses the Georgia Tech Internet Topology Model (GT-ITM) for generating problem instances. It has been used to validate the D-ViNE and R-ViNE algorithms.

MAPF

In the MAPF problem, we are given a directed or undirected graph $G = (V, E)$ and a set of K agents $1, 2 \dots K$. Each agent j has a unique start vertex $s^j \in V$ and a unique goal vertex $g^j \in V$. At each time step, each agent can either move to a neighboring vertex or wait at its current vertex, both with unit cost. A solution of a MAPF problem instance is a set of feasible paths, one path $\{s_0^j, s_1^j \dots s_{T_j}^j, s_{T_j+1}^j \dots\}$ for each agent $j \in \{1, 2 \dots K\}$, such that no two paths conflict. A path for agent j is feasible if and only if, (a) it starts at the start vertex of agent j, i.e., $s_0^j = s^j$; (b) it ends at the goal vertex of agent j and remains there, i.e., there exists a smallest T_j such that $s_{T_j}^j = g^j$ and, for each $t > T_j$, $s_t^j = g^j$; and (c) every action is a legal move or wait action, i.e., for all $t \in \{0, 1 \dots T_j - 1\}, (s_t^j, s_{t+1}^j) \in E$ or $s_t^j = s_{t+1}^j$. A conflict between the paths of agents j and k is either a vertex conflict (j, k, s, t), i.e., $s = s_t^j = s_t^k$, or an edge conflict (j, k, s_1, s_2, t), i.e., $s_1 = s_t^j = s_{t+1}^k$ and $s_2 = s_{t+1}^j = s_t^k$. The travel cost of agent j, (also called the cost of its path) is the number of time steps T_j until it reaches its goal vertex and stays there. One common objective is to minimize the solution cost in the form of the flow time, given as the sum of the travel costs of all agents (Yu and LaValle 2013; Sharon et al. 2015). A variety of MAPF solvers have been developed by different research groups, see (Wagner 2015) and (Hoenig et al. 2016) for overviews.

CBS

CBS (Sharon et al. 2015) is an optimal MAPF solver. It performs high-level and low-level searches. Each high-level node (also called CT node) contains a set of constraints and, for each agent, a feasible path that respects those constraints. The root CT node has no constraints. The high-level search of CBS is a best-first search that uses the costs of the CT nodes as their f-values. The cost of a CT node is the sum of the travel costs along its paths. When CBS expands a CT node N, it checks whether the node is a goal node. A CT node is a goal node if and only if none of its paths conflict. If N is a goal node, then CBS terminates successfully and outputs the paths in N as a solution. Thus, the fewer conflicts there are in CT nodes, the faster CBS terminates. Otherwise, at least two paths conflict. CBS chooses a conflict to resolve and generates two CT child nodes of N, called N_1 and N_2. Both N_1 and N_2 inherit the constraints of N. If the chosen conflict is a vertex conflict (j, k, s, t), then CBS adds the vertex constraint (j, s, t) to N_1 (that prohibits agent j from occupying vertex s at time step t) and the vertex constraint (k, s, t) to N_2. If the chosen conflict is an edge conflict (j, k, s_1, s_2, t), then CBS adds the edge constraint (j, s_1, s_2, t) to N_1 (that prohibits agent j from moving from vertex s_1 to vertex s_2 between time steps t and $t + 1$) and the edge constraint (k, s_2, s_1, t) to N_2. During the generation of the CT node N, CBS performs a low-level search for the agent i affected by the added constraint. The low-level search for agent i is a best-first A* search that ignores all other agents and finds a minimum-cost path from the start vertex of agent i to its goal vertex that is both feasible and respects the constraints of the CT node N that involve agent i. Gordon, Filmus, and Salzman (2021) present a complexity analysis of CBS.

Enhanced CBS (ECBS) (Barer et al. 2014) is a CBS-based MAPF solver that invokes the power of focal search. ECBS(w) is a w-suboptimal variant of CBS whose high-level and low-level searches are focal searches rather than best-first searches. ECBS(w) is w-suboptimal since it guarantees a solution with a cost that is no larger than a constant suboptimality factor w times the optimal solution cost. A focal search, like A* search, uses a list *OPEN* whose nodes n are sorted in increasing order of their f-values $f(n) = g(n) + h(n)$, where $h(n)$ are the primary heuristic values. Unlike A* search, a focal search with suboptimality factor w also uses a list *FOCAL* of all nodes currently in *OPEN* whose f-values are no larger than w times the currently smallest f-value in *OPEN*. The nodes in *FOCAL* are sorted in increasing order of their secondary heuristic values. While A* search expands a node in *OPEN* with the smallest f-value, a focal search expands a node in *FOCAL* with the smallest secondary heuristic value. Thus, the secondary heuristic values should favor a node in *FOCAL* that is close to a goal node to speed up the search and thus exploit the leeway afforded by the suboptimality factor that A* search does not have available. If the primary heuristic values are admissible (never overestimate the cost of reaching the goal), then a focal search is w-suboptimal. The secondary heuristic values can be inadmissible (may overestimate the cost of reaching the goal). The high-level and low-level searches of ECBS(w) are focal searches. During the generation of a CT node N, ECBS(w) performs a low-level focal search with $OPEN^i(N)$ and $FOCAL^i(N)$ for the agent i affected by the added constraint. The high-level and low-level focal searches of ECBS(w) use measures related to the number of conflicts as secondary heuristic values. Thus, ECBS(w) with a reasonably small value of $w > 1$ can expand the CT nodes with fewer conflicts than the CT nodes chosen by CBS. This often makes ECBS(w) find a solution CT node faster.

VNE: Problem Formulation and Example

In this paper, we focus on the core VNE problem with geographical constraints, as proposed in (Chowdhury, Rahman, and Boutaba 2009). The SN is represented as an undirected graph $G^s = (V^s, E^s, A_V^s, A_E^s)$. V^s is the set of SN vertices, E^s is the set of SN edges, A_V^s is a mapping from SN vertices to their attributes, and A_E^s is a mapping from SN edges to their attributes. The attributes of an SN vertex $v^s \in V^s$ are its CPU capacity $\text{CPU}(v^s)$ and its location $\text{LOC}(v^s)$. The attribute of an SN edge $e^s \in E^s$ is its bandwidth capacity $\text{BW}(e^s)$. An SN path is a path in G^s.

A VNR is characterized by an undirected graph $G^r =$

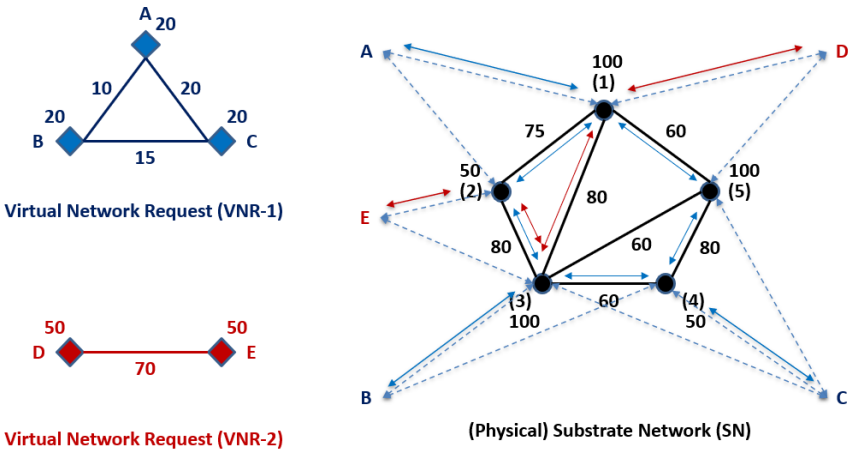

Figure 1: Shows two VNRs, VNR-1 (blue) and VNR-2 (red), on the left side and the (physical) SN (black), with additional fictitious vertices and edges, on the right side. The non-negative numbers annotating the VNR vertices represent their CPU requirements. The non-negative numbers annotating the VNR edges represent their bandwidth requirements. The SN has vertices (1), (2), (3), (4), and (5). The non-negative numbers annotating the SN vertices represent their CPU capacities. The non-negative numbers annotating the SN edges represent their bandwidth capacities. There is a fictitious vertex on the right side for each VNR vertex. Each such VNR vertex is connected to the SN vertices that it can possibly be mapped to via fictitious edges, representing the geographical constraints. A feasible VNE solution, mapping each VNR vertex to exactly one SN vertex and mapping each VNR edge to exactly one SN path, is shown with solid line segments with arrowheads at both ends. That is, the VNR vertices A, B, C, D, and E are mapped to the SN vertices (1), (3), (4), (1), and (2), respectively; and the VNR edges A-B, B-C, A-C, and D-E are mapped to the SN paths (1)-(2)-(3), (3)-(4), (1)-(5)-(4), and (1)-(3)-(2), respectively.

(V^r, E^r, C^r_V, C^r_E). 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 popularly considered demands of a VNR vertex $v^r \in V^r$ are its CPU requirements $\text{CPU}(v^r)$, its location $\text{LOC}(v^r)$, and the maximum allowed distance $D(v^r)$ from its location to the location of the mapped SN vertex. The popularly considered demand of a VNR edge $e^r \in E^r$ is its bandwidth requirement $\text{BW}(e^r)$.

Given a VNR G^r, a feasible VNE is a mapping $\text{VNE}(\cdot)$ of VNR vertices to SN vertices and VNR edges to SN paths such that

1. each VNR vertex $v^r \in V^r$ is mapped to a unique SN vertex $\text{VNE}(v^r) \in V^s$,
2. no two VNR vertices $v^r_i \in V^r$ and $v^r_j \in V^r$ from the same VNR are mapped to the same SN vertex $v^s \in V^s$,
3. for any VNR vertex $v^r \in V^r$:
   $\text{CPU}(v^r) \leq \text{CPU}(\text{VNE}(v^r))$ and
   $\text{GEODIST}(\text{LOC}(v^r), \text{LOC}(\text{VNE}(v^r))) \leq D(v^r)$,
   where $\text{GEODIST}(\cdot,\cdot)$ is the geographical distance function between two locations,
4. each VNR edge $(v^r_i, v^r_j) \in E^r$ is mapped to an SN path $\text{VNE}((v^r_i, v^r_j))$ from $\text{VNE}(v^r_i)$ to $\text{VNE}(v^r_j)$ in G^s, and
5. for any SN edge $e^s \in E^s$:
   $$\sum_{e^r \in E^r:\ e^s \in \text{VNE}(e^r)} \text{BW}(e^r) \leq \text{BW}(e^s).$$

The revenue of a VNE mapping is defined as $\sum_{v^r \in V^r} \text{CPU}(v^r) + \sum_{e^r \in E^r} \text{BW}(e^r)$, and its cost is defined as $\sum_{v^r \in V^r} \text{CPU}(v^r) + \sum_{e^r \in E^r} \sum_{e^s \in \text{VNE}(e^r)} \text{BW}(e^r)$. The cost of embedding a VNR is larger or equal to its revenue.

In this paper, the VNE problem is to find a feasible VNE mapping, if one exists, that minimizes the cost of the mapping since the revenues of all feasible VNE mappings are identical. Figure 1 shows an example of mapping two VNRs to an SN. One commonly used way of formulating the VNE problem as a "path coordination" problem is to represent each VNR vertex $v^r \in V^r$ as a fictitious vertex $v^f \in V^f$ in a graphical representation of the SN (Chowdhury, Rahman, and Boutaba 2009). Each fictitious vertex $v^f \in V^f$ inherits all attributes of the VNR vertex v^r, such as $\text{CPU}(v^r)$, $\text{LOC}(v^r)$, and $D(v^r)$. Each v^f is connected via fictitious edges $(v^f, v^s) \in E^f$ to all SN vertices $v^s \in V^s$ with $\text{GEODIST}(\text{LOC}(v^f), \text{LOC}(v^s)) \leq D(v^f)$. Each fictitious edge has infinite bandwidth capacity and represents the mapping of a VNR vertex to an SN vertex. This new graph G^m is called the augmented graph, and a VNE mapping is a set of paths on it. Its vertices are $V^s \cup V^f$, and its edges are $E^s \cup E^f$. Each path on G^m connects two fictitious vertices with two fictitious edges (the mappings of a VNR vertex to an SN vertex) and SN edges (the mapping of a VNR edge to an SN path). Such a path cannot pass through any fictitious vertices between the start and goal fictitious vertices. The VNE problem can then be interpreted as a path coordination problem with CPU and bandwidth constraints.

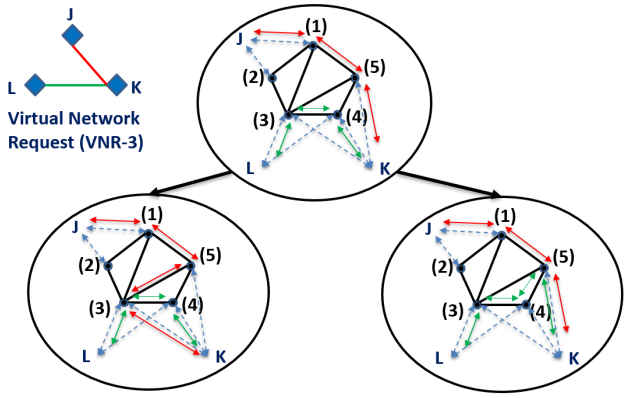

Figure 2: Shows VNR-3 and an example of CT nodes for mapping VNR-3 to the SN graph from Figure 1. The root CT node contains a VNE mapping with a vertex conflict since fictitious vertex K is mapped to two SN vertices (4) and (5). To resolve this vertex conflict, we create two child CT nodes. The left-child CT node has a new constraint stopping fictitious vertex K from being mapped to (5), and the red path is rerouted to accommodate the new constraint. Similarly, the right-child CT node has a new constraint stopping fictitious vertex K from being mapped to (4), and the green path is rerouted. The left-child CT node still has a vertex conflict since K is mapped to (3) and (4), and this vertex conflict can be resolved by expanding this CT node in the same manner. The right-child node is a goal CT node with a feasible VNE mapping since it has no conflicts.

Of course, additional constraints should be introduced to enforce that each fictitious vertex chooses exactly one fictitious edge incident on it for all its SN paths. Figure 1 illustrates this mechanism.

As shown in the definition of the cost of a VNE mapping, the amount of allocated CPU capacities is identical for all feasible VNE mappings, and the cost of mapping each VNR edge $e^r \in E^r$ is the bandwidth requirement $\text{BW}(e^r)$ multiplied by the mapped SN path length. Thus, minimizing the cost of a VNE mapping is the same as minimizing the sum of the lengths of paths on G^m (corresponds to the sum of costs in CBS).

VNE-CBS

We adapt the CBS framework to the VNE problem. A CT node N now contains a VNE mapping $N.mapping$, and the cost of the CT node $N.cost$ becomes the cost of its VNE mapping. A kind of "vertex conflict" arises when a VNR vertex v^r is mapped to two different SN vertices v^s_1 and v^s_2. To resolve it, we create two child CT nodes, one with a new constraint (v^r, v^s_1) stopping v^r from being mapped to v^s_1 and the other one (v^r, v^s_2) stopping v^r from being mapped to v^s_2. Figure 2 presents an example of resolving a vertex conflict. Another kind of "vertex conflict" arises when two VNR vertices v^r_1 and v^r_2 from the same VNR are mapped to the same SN vertex v^s. To resolve it, we create two child CT nodes, one with a new constraint (v^r_1, v^s) stopping v^r_1 from being mapped to v^s and the other one (v^r_2, v^s) stopping v^r_2 from being mapped to v_s. A "bandwidth capacity conflict" arises when a VNR edge e^r utilizes an SN edge e^s that does not have sufficient bandwidth capacity to accommodate $\text{BW}(e^r)$. To resolve it, we take all VNR edges in $\{e^r \in E^r : e^s \in \text{VNE}(e^r)\}$ and create a child CT node for each such VNR edge with a new constraint that stops it from utilizing e^s. We define $N.constraints$ as the set of constraints imposed for resolving conflicts. The geographical constraints are enforced while constructing the augmented graph G^m and therefore not in $N.constraints$.

Algorithm 1: VNE-CBS

```
1: Input: G^s, G^r, w
2: G^m ← augmented graph for G^s and G^r
3: N_R ← empty CT node
4: N_R.constraints ← ∅
5: N_R.mapping ← low-level paths found in G^m for all
   VNR edges using Algorithm 2
6: N_R.cost ← cost(N_R.mapping)
7: N_R.num_conf ← number of conflicts in N_R.mapping
8: OPEN = FOCAL = {N_R}
9: while FOCAL ≠ ∅ do
10:     cost_old ← OPEN.top().cost
11:     N_T ← FOCAL.top()
12:     Remove N_T from OPEN and FOCAL
13:     if N_T.num_conf = 0 then
14:         return N_T.mapping as solution
15:     Conf ← first conflict found in N_T.mapping
16:     Cons ← generate constraints for resolving Conf
17:     for c ∈ Cons do
18:         N_C ← copy of N_T
19:         Add c to N_C.constraints
20:         Update low-level paths in N_C.mapping using
            Algorithm 2 to accommodate constraint c
21:         if update successful then
22:             N_C.cost ← cost(N_C.mapping)
23:             N_C.num_conf ← number of conflicts
                in N_C.mapping
24:             OPEN ← OPEN ∪ {N_C}
25:             if N_C.cost ≤ w · OPEN.top().cost then
26:                 FOCAL ← FOCAL ∪ {N_C}
27:     cost_new ← OPEN.top().cost
28:     for N ∈ OPEN do
29:         if w · cost_old < N.cost ≤ w · cost_new then
30:             FOCAL ← FOCAL ∪ {N}
31: return "No Solution"
```

The VNE-CBS algorithm uses a high-level focal search with suboptimality factor $w \geq 1$, see Algorithm 1. VNE-CBS takes three inputs: an SN graph G^s, a VNR graph G^r, and a suboptimality factor w for minimizing the cost. On Line 2, it uses G^s and G^r to create an augmented graph G^m containing the fictitious vertices and edges, as described in Figure 1. On Lines 3-7, it initializes the root CT node N_R. On Line 5, the low-level search finds a path from v^f_1 to v^f_2 on G^m for each VNR edge (v^r_1, v^r_2) where v^f_1 and v^f_2 are the

Algorithm 2: Low-Level Search

```
1: Input: v_1^r, v_2^r, G^m, N.constraints, N.mapping
2: v_1^f, v_2^f ← fictitious vertices for v_1^r and v_2^r
3: n_R ← v_1^f
4: n_R.g ← 0
5: OPEN ← {n_R}
6: CLOSED ← ∅
7: while OPEN ≠ ∅ do
8:     n_T ← OPEN.top()
9:     Remove n_T from OPEN
10:    if n_T = v_2^f and par(par(n_T)) ≠ v_1^f then
11:        return make_path(n_T) as path
12:    Neighbors ← valid_neighbors(n_T)
13:    for n ∈ Neighbors and n ∉ CLOSED do
14:        n.g ← n_T.g + 1
15:        OPEN ← OPEN ∪ {n}
16:    CLOSED ← CLOSED ∪ {n_T}
17: return failure
```

fictitious vertices for v_1^r and v_2^r respectively. $N_R.mapping$ is this set of paths, $N_R.cost$ is the cost of the mapping, and $N_R.num\_conf$ is the number of violated constraints (also called conflicts) corresponding to the number of conflicts in CBS. On Line 8, VNE-CBS inserts N_R into priority queues *OPEN* and *FOCAL*. Here, the cost is used as the primary heuristic, and the CT node with the smallest cost is maintained on top of the priority queue *OPEN*. Similarly, the number of conflicts is used as the secondary heuristic, and the CT node with the smallest number of conflicts is maintained on top of the priority queue *FOCAL*.

VNE-CBS then expands CT nodes until either a feasible VNE mapping is found or *FOCAL* is empty. On Lines 10-12, it retrieves the node with the smallest number of conflicts from *FOCAL* and removes it from *OPEN* and *FOCAL*, following the standard focal search procedure. On Lines 13-14, VNE-CBS returns a feasible mapping if no conflicts exist in $N_R.mapping$. Otherwise, on Lines 15-16, it finds a conflict in $N_R.mapping$ and generates constraints *Cons* to resolve it. It first checks the mapping of each VNR vertex and then the mapping of each VNR edge. In case of the existence of multiple conflicts, it prefers vertex conflicts over the other conflicts and generates constraints for the first conflict in the order of appearance.

On Lines 17-19, VNE-CBS creates a child CT node N_C for each constraint $c \in$ *Cons* by making a copy of CT node N_T and adding the new constraint c to $N_C.constraints$. On Line 20, it updates $N_C.mapping$ to accommodate the added constraint by recomputing the affected paths with the low-level search procedure. If the path update succeeds, on Lines 22-24, VNE-CBS calculates the cost and number of conflicts for child node N_C and inserts N_C into *OPEN*. On Lines 25-26, following the standard focal search procedure, it inserts child node N_C into *FOCAL* if $N_C.cost$ is within w times the cost of the top node in *OPEN*. On Lines 27-30, VNE-CBS updates *FOCAL* in case the cost of the top node in *OPEN* has increased as a result of the previous operations.

We now present the low-level search procedure, see Algorithm 2, for finding a path from one fictitious vertex v_1^f to another fictitious vertex v_2^f in G^m in the presence of the constraints $N.constraints$ where N is the CT node associated with this low-level search. The fictitious vertices v_1^f and v_2^f represent the VNR vertices v_1^r and v_2^r, respectively. Each fictitious vertex $v^f \in V^f$ inherits all attributes of the VNR vertex v^r, such that $\text{CPU}(v^f) = \text{CPU}(v^r)$, $\text{LOC}(v^f) = \text{LOC}(v^r)$, and $D(v^f) = D(v^r)$. It starts from v_1^f and performs a best-first search until it finds a path to v_2^f of minimum path length.

When expanding a node n_T, Algorithm 2 considers it a goal node if $n_T = v_2^f$ and the parent ($\texttt{par}(\cdot)$) of the parent of n_T is not v_1^f, see Line 10. This prevents the case where a path of length 3, i.e., $\langle v_1^f, v^s, v_2^f \rangle$, gets returned as a solution path, since such a path maps two VNR vertices from the same VNR to the same SN vertex (resulting in a vertex conflict). On Line 12, the function $\texttt{valid\_neighbors}(\cdot)$ finds the valid neighbors of n_T. When $n_T = v_1^f$, it finds those SN vertices v^s that satisfy $\text{CPU}(v^s) \geq \text{CPU}(v_1^f)$. When n_T is an SN vertex v^s and v_2^f is one of the neighbors of n_T, the function includes v_2^f if $\text{CPU}(v^s) \geq \text{CPU}(v_2^f)$. The geographical distance requirements of v_1^f and v_2^f are enforced during the construction of the augmented graph. When n_T is an SN vertex, the function excludes the fictitious vertices that are not v_2^f, since a path is not allowed to go through other fictitious vertices, and finds those SN vertices v^s that satisfy $\text{BW}(e^s) \geq \text{BW}(e^r)$, where $e^s = (n_T, v^s)$ and $e^r = (v_1^r, v_2^r)$. The function also excludes the neighboring vertices that are ruled out by the constraints $N.constraints$. On Lines 13-15, Algorithm 2 generates a child node for each valid neighbor n.

Algorithm 2 uses a tie-breaking rule when selecting the node with the minimum g-value from *OPEN*. When mapping v_1^f and v_2^f to SN vertices, it goes through the mapping of the other VNR vertices in $N.mapping$ and counts the number of different VNR vertices mapped to each SN vertex. If more than one SN vertices have the same g-value, it prefers the SN vertex that is mapped to different VNR vertices the fewest number of times. Breaking ties in this way reduces the number of potential vertex conflicts that the returned solution path may cause in the VNE mapping. As a result, it reduces the necessary number of CT node expansions for finding a feasible VNE mapping.[1]

Since the VNE problem is NP-hard, in the worst case, VNE-CBS expends an amount of work that grows exponentially in the number of conflicts encountered. An upper bound on the complexity of VNE-CBS is exponential both in the size of the VNR and the size of the SN.

Theorem 1. *VNE-CBS is complete and w-suboptimal. It is complete and optimal for $w = 1$.*

Proof. The formal proof will be available on http://idm-lab.org in a longer version of the paper. □

[1] VNE-CBS algorithms that use this tie-breaking rule will be tagged with the label "tie" in the next section.

With a few modifications of VNE-CBS, it has the potential to map T VNRs $G_1^r, G_2^r \dots G_T^r$ at once. Here, we describe the idea of the modifications. The first step is to merge all VNRs into a single VNR and mark VNR vertices and edges from each VNR G_k^r with the VNR index k. On Line 15 of Algorithm 1, a vertex conflict does not arise if two VNR vertices with different VNR indices are mapped to the same SN vertex. Then we introduce a new kind of conflict "CPU capacity conflict" that arises when a VNR vertex is mapped to an SN vertex v^s such that $\sum_{v^r \in V_1^r \cup V_2^r \dots V_T^r:\ \text{VNE}(v^r)=v^s} \text{CPU}(v^r) > \text{CPU}(v^s)$. To resolve it, we take all VNR vertices in $\{v^r \in V_1^r \cup V_2^r \dots V_T^r : \text{VNE}(v^r) = v^s\}$ and create a child CT node for each such VNR vertex with a new constraint that stops it from being mapped to v^s.

Experiments

In this section, we present experimental results comparing our VNE-CBS against D-ViNE, R-ViNE, G-SP, and G-MCF, four popularly used VNE algorithms for the defined core VNE problem. New VNE algorithms are commonly compared to these four algorithms in the VNE literature (Zheng et al. 2017; Yan et al. 2020). As mentioned before, the deterministic VN embedding algorithm D-ViNE and the randomized VN embedding algorithm R-ViNE are rounding algorithms that heuristically retrieve a solution from a fractional solution of an LP problem that can be obtained in polynomial time. The LP problem comes from a relaxation of the MILP formulation of the VNE problem (Chowdhury, Rahman, and Boutaba 2009). Both algorithms produce path-splitting solutions. G-SP (Greedy-Shortest Path) is a greedy algorithm that uses shortest path algorithms for mapping VNR edges (Zhu and Ammar 2006). G-MCF (Greedy-Multi-Commodity Flow) is a greedy algorithm that uses multi-commodity flow algorithms for mapping VNR edges (Yu et al. 2008). We implemented our CBS-based algorithms in C++ [2]. The vanilla versions of our VNE-CBS algorithm, with and without the tie-breaking rule, use $w = 1.00$ and are labeled VNE-CBS-w1.00-tie and VNE-CBS-w1.00, respectively. They are complete and optimal. The suboptimal variants, with and without the tie-breaking rules, use $w = 1.01$, 1.05, 1.10, 1.50, and 2.00 in the experiments. We conducted experiments with the VNE simulator ViNEYard (Chowdhury, Rahman, and Boutaba 2012). All experiments were run on an AWS machine with 8 CPUs and 16GB RAM.

We used a standard methodology from the VNE literature to generate VNE problem instances via Waxman graphs. Waxman graphs (Waxman 1988) are frequently chosen in simulations as topologies resembling communication networks. The SN topologies in our experiments are randomly generated Waxman graphs with parameter values $\alpha = 0.5$ and $\beta = 0.2$. We generated 5 SNs. Each SN has 100 vertices in a 50×50 grid space. The CPU and bandwidth capacities of the SN vertices and edges are real numbers generated uniformly at random from the interval $[50, 100]$.

The VNR topologies are also generated using this Waxman method. Following the experimental setup used in previous works (Chowdhury, Rahman, and Boutaba 2009; Yu et al. 2008), we set the number of vertices in each VNR to be an integer drawn uniformly at random from the interval $[2, 10]$. The VNR vertices are located in the same 50×50 grid space as the SN vertices. Geographical constraints with threshold distance 15 (measured as Euclidean distance) are used to specify the mappable SN vertices for each VNR vertex. The CPU requirements of the VNR vertices are real numbers drawn uniformly at random from the interval $[0, 20]$. The bandwidth requirements of the VNR edges are real numbers drawn uniformly at random from the interval $[0, 50]$. VNRs generated in this way are used in Tables 1 and 5. We generated $2,000$ VNRs.

In order to stress test the various VNE algorithms, we generated VNRs slightly differently in Tables 2, 3, and 4. Table 2 sets the number of vertices in each VNR to be an integer drawn uniformly at random from the interval $[2, 20]$ (instead of the interval $[2, 10]$). Table 3 sets the CPU requirements of the VNR vertices to be real numbers drawn uniformly at random from the interval $[0, 50]$ (instead of the interval $[0, 20]$). Table 4 sets the bandwidth requirements of the VNR edges to be real numbers drawn uniformly at random from the interval $[0, 80]$ (instead of the interval $[0, 50]$). Tables 2, 3, and 4 increase the difficulty of mapping VNRs onto the SNs. They provide insights into the efficiency and effectiveness of the various VNE algorithms for increasing demand levels.

Tables 1, 2, 3, and 4 are used to test the VNE algorithms in an offline setting. Figures 3, 4, 5, and 6 present the results of the suboptimal variants of VNE-CBS with different w for the experimental settings in Tables 1, 2, 3, and 4, respectively. In the figures, the data points at $w = 1.00$ correspond to the performance of VNE-CBS-w1.00 and VNE-CBS-w1.00-tie. In each run, we mapped each of the $2,000$ VNRs to each of the 5 SNs, and therefore there were $10,000$ VNE instances solved independently. The tables report the results averaged over 5 runs for the number of solved instances (# Solved), the number of instances for which the algorithm timed out after 60 seconds (# Timeout), and the number of instances for which the algorithm conclusively returns the non-existence of solutions (# No Solution). We use "-" in the # No Solution columns if an algorithm cannot return the non-existence of solutions. All variants of VNE-CBS are complete, i.e., # Solved + # Timeout + # No Solution is always equal to $10,000$. On the other hand, none of the other state-of-the-art algorithms are complete, i.e., as in Table 4, # Solved + # Timeout + # No Solution is not always equal to $10,000$. The tables and figures also report results averaged over all runs and all VNRs that are successfully embedded by all algorithms. The reported metrics include the average runtime, average cost, and the average number of CBS nodes expanded in the high-level search (when relevant). All variants of VNE-CBS significantly outperform D-ViNE and R-ViNE in terms of runtime. They are also competitive with G-MCF in terms of runtime, frequently outperforming it. In addition, all versions of VNE-CBS sig-

[2] https://github.com/YiZ7699/VNE-CBS

Algorithm	# Solved	# Timeout	# No Solution	Avg Runtime (s)	Avg Cost	Avg # CT Nodes
D-ViNE	10,000	0	-	1.885	223.935	-
R-ViNE	10,000	0	-	1.904	224.765	-
G-SP	10,000	0	-	0.022	273.085	-
G-MCF	10,000	0	-	0.495	234.178	-
VNE-CBS-w1.00	9,432	568	0	0.829	141.613	130.899
VNE-CBS-w1.00-tie	9,616	384	0	0.627	141.613	91.400

Table 1: Offline Setting: number of VNR vertices $\in_u [2, 10]$; VNR vertex CPU requirement $\in_u [0, 20]$; VNR edge bandwidth requirement $\in_u [0, 50]$.

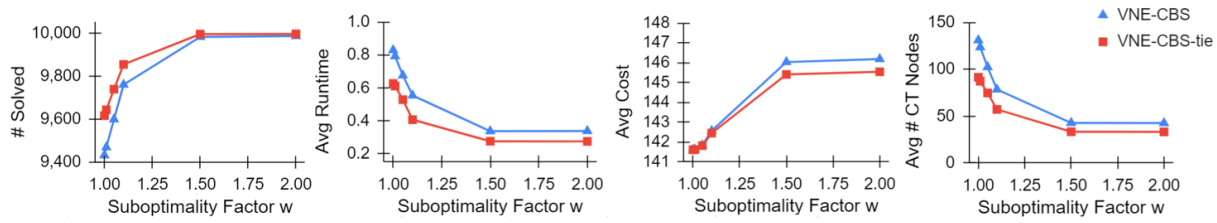

Figure 3: The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 1.

Algorithm	# Solved	# Timeout	# No Solution	Avg Runtime (s)	Avg Cost	Avg # CT Nodes
D-ViNE	9,996	0	-	3.479	305.610	-
R-ViNE	9,996	0	-	3.475	304.969	-
G-SP	9,982	0	-	0.030	369.374	-
G-MCF	9,996	0	-	0.824	314.569	-
VNE-CBS-w1.00	5,334	4,666	0	2.043	188.664	192.310
VNE-CBS-w1.00-tie	5,813	4,187	0	1.719	188.664	166.603

Table 2: Offline Setting: number of VNR vertices $\in_u [2, 20]$; VNR vertex CPU requirement $\in_u [0, 20]$; VNR edge bandwidth requirement $\in_u [0, 50]$.

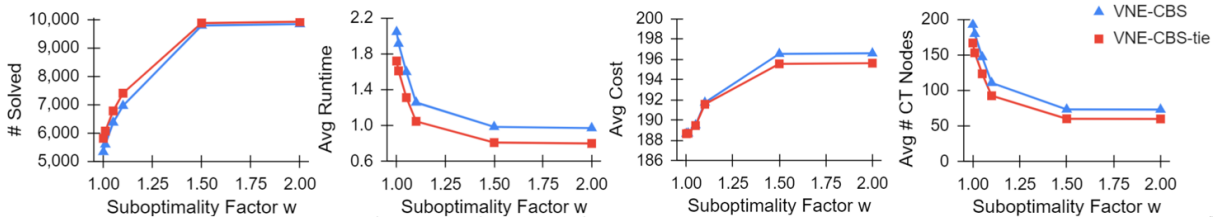

Figure 4: The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 2.

Algorithm	# Solved	# Timeout	# No Solution	Avg Runtime (s)	Avg Cost	Avg # CT Nodes
D-ViNE	10,000	0	-	2.294	296.922	-
R-ViNE	10,000	0	-	2.293	297.112	-
G-SP	10,000	0	-	0.026	344.033	-
G-MCF	10,000	0	-	0.717	306.871	-
VNE-CBS-w1.00	9,378	622	0	0.873	217.628	109.340
VNE-CBS-w1.00-tie	9,553	447	0	0.658	217.628	78.575

Table 3: Offline Setting: number of VNR vertices $\in_u [2, 10]$; VNR vertex CPU requirement $\in_u [0, 50]$; VNR edge bandwidth requirement $\in_u [0, 50]$.

nificantly outperform all other algorithms in terms of cost. However, the vanilla version of VNE-CBS solves fewer instances compared to the four baseline algorithms since it explores all possible mappings until it finds a VNE mapping of minimum cost. As shown in the figures, compared to VNE-CBS-w1.00, the suboptimal variants with larger w solve significantly more instances, have significantly better runtimes, and expand significantly fewer CT nodes. The same trends hold for VNE-CBS-w1.00-tie and the suboptimal variants of it with larger w. Moreover, the VNE-CBS versions with the tie-breaking rule are better than the corresponding vanilla versions.

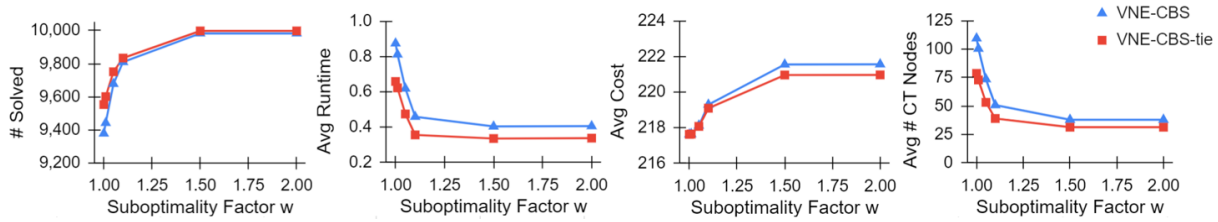

Figure 5: The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 3.

Algorithm	# Solved	# Timeout	# No Solution	Avg Runtime (s)	Avg Cost	Avg # CT Nodes
D-ViNE	9,997	0	-	1.910	319.102	-
R-ViNE	9,996	0	-	1.896	320.026	-
G-SP	9,966	0	-	0.026	428.909	-
G-MCF	9,997	0	-	0.647	333.854	-
VNE-CBS-w1.00	9,213	786	1	0.779	188.019	147.290
VNE-CBS-w1.00-tie	9,423	576	1	0.624	188.015	107.690

Table 4: Offline Setting: number of VNR vertices $\in_u [2, 10]$; VNR vertex CPU requirement $\in_u [0, 20]$; VNR edge bandwidth requirement $\in_u [0, 80]$.

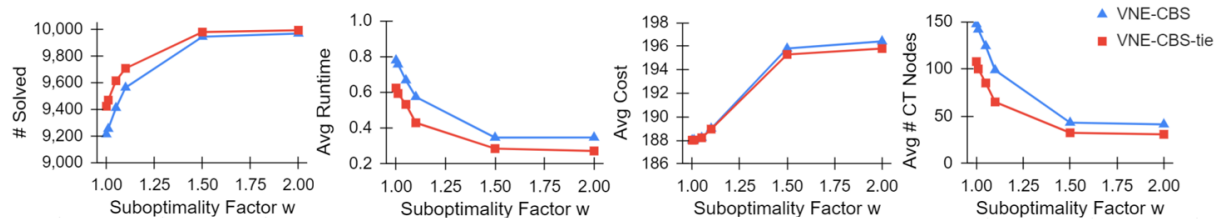

Figure 6: The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 4.

Algorithm	Avg Acceptance Ratio	Avg Revenue	Avg Cost	Avg Revenue/Avg Cost	Avg Runtime (s)
D-ViNE	0.987	293,680.8	491,697.3	0.597	3,852.128
R-ViNE	0.993	295,975.0	490,550.8	0.603	3,852.340
G-SP	0.967	285,677.1	604,714.6	0.472	39.632
G-MCF	0.969	286,858.4	494,049.2	0.581	1,043.922
VNE-CBS-w1.00	0.953	243,231.1	243,322.1	0.999	7,619.500
VNE-CBS-w1.00-tie	0.959	276,522.1	276,684.1	0.999	6,747.309

Table 5: Online Setting: number of VNR vertices $\in_u [2, 10]$; VNR vertex CPU requirement $\in_u [0, 20]$; VNR edge bandwidth requirement $\in_u [0, 50]$.

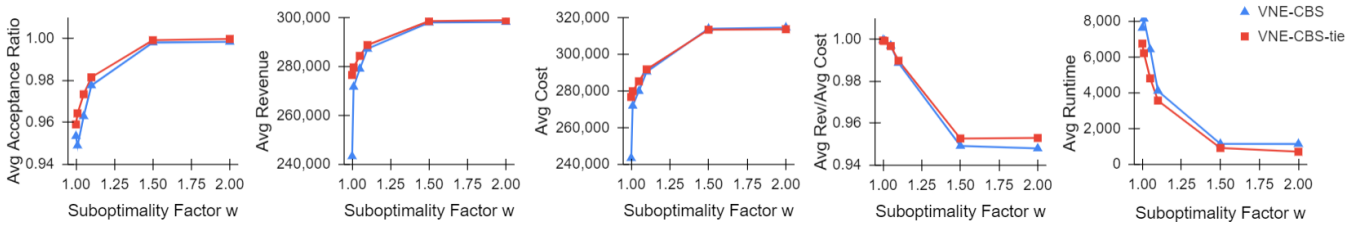

Figure 7: The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 5.

Table 5 is used to test the VNE algorithms in an online setting, and Figure 7 presents the performance of the suboptimal variants of VNE-CBS with different w for the same experimental setting. Here, VNRs arrive dynamically, and each successfully mapped VNR holds the CPU and bandwidth capacities allocated to it on the SN until it departs at the end of its lifetime. In this paper, mapping each VNR does not reconfigure the mapping of the previous VNRs. If an algorithm fails to map a VNR within the time limit (60 seconds), it rejects the VNR and embeds the next one. In the online setting used for Table 5, VNRs arrive according to a Poisson process at an average rate of 4 VNRs per 100 time

steps. The lifetime of each VNR is generated from an exponential distribution with an average of $1,000$ time steps. In each run, corresponding to each of the 5 SNs, the $2,000$ generated VNRs arrive and depart in sequence. Table 5 and Figure 7 report the results averaged over the 5 runs. The average revenue and average cost are the sum of the revenue and cost of all successfully mapped VNRs averaged over the 5 runs. The average runtime is the runtime of embedding the $2,000$ VNRs averaged over the 5 runs. We observe that all VNE algorithms have a high average acceptance ratio. Consequently, they have very similar average revenues. But they differ remarkably in their average cost, and, therefore, also in their ratio of average revenue to average cost. The various VNE algorithms also differ in their average runtime. All variants of VNE-CBS significantly outperform the baseline algorithms in terms of average cost and the ratio of average revenue to average cost. VNE-CBS-w1.00 and VNE-CBS-w1.00-tie have the best performance in terms of the ratio of average revenue to average cost since they are optimal. Their runtime and average acceptance ratios are lower than those of the other variants and the baseline algorithms since finding an optimal VNE is time-consuming and many instances time out. In addition, VNE-CBS-tie with $w = 2.00$ has an average runtime of 698.742 seconds and is significantly faster than all other non-greedy algorithms.

Conclusions and Future Work

In this paper, we presented VNE-CBS, a novel CBS-based solver for the VNE problem. The VNE problem is an NP-hard problem in resource management. It models the fundamental combinatorics in emerging network virtualization architectures, by which service providers will be able to create many heterogeneous virtual networks and offer customized end-to-end services by leasing shared resources from infrastructure providers. The VNE problem involves the proper allocation of CPU and bandwidth capacities in a physical SN to meet the demands of multiple VNs.

VNE-CBS is inspired by the success of the CBS framework in the MAPF domain, since both the VNE and MAPF problems aim to find a set of paths in a graph while satisfying certain constraints. While VNE-CBS exploits the similarities between the VNE problem and the MAPF problem, it also addresses the subtle differences and unique challenges in applying the CBS framework to the VNE problem. We showed that, unlike many existing algorithms for solving the VNE problem, VNE-CBS is complete and optimal. We also developed a bounded-suboptimal version of VNE-CBS that trades off optimality for increased efficiency. Finally, through a wide range of experiments, we showed that our CBS-based algorithms significantly outperform state-of-the-art methods on various benchmark instances, for both the offline and online versions of the VNE problem.

While there are many variants of the VNE problem, depending on whether path-splitting is allowed, whether there are additional considerations on sharing bandwidth beyond just capacity constraints, and so forth, we focused on a core version of the VNE problem as a starting point to (a) peel the complexities of the richer versions of the problem and (b) import AI techniques to solve problems in 5G technologies and network slicing. Overall, our methodology paves the way for overcoming Internet ossification via heuristic search methods. In principle, our CBS-based methodology can be applied in any context where modularization of services is required. In future work, we will generalize VNE-CBS to solve other variants of the VNE problem using algorithmic ideas similar to those developed in the MAPF literature.

Acknowledgements

This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government.

References

Andersen, D. G. 2002. *Theoretical Approaches to Node Assignment*. Ph.D. thesis, Carnegie Mellon University.

Anderson, T. E.; Peterson, L. L.; Shenker, S.; and Turner, J. S. 2005. Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4): 34–41.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *International Symposium on Combinatorial Search*, 19–27.

Cao, H.; Wu, S.; Hu, Y.; Liu, Y.; and Yang, L. 2019. A Survey of Embedding Algorithm for Virtual Network Embedding. *China Communications*, 16(12): 1–33.

Cheng, X.; Su, S.; Zhang, Z.; Wang, H.; Yang, F.; Luo, Y.; and Wang, J. 2011. Virtual Network Embedding Through Topology-Aware Node Ranking. *Computer Communication Review*, 41(2): 38–47.

Choudhury, S.; Solovey, K.; Kochenderfer, M. J.; and Pavone, M. 2020. Efficient Large-Scale Multi-Drone Delivery Using Transit Networks. In *IEEE International Conference on Robotics and Automation*, 4543–4550.

Chowdhury, M.; and Boutaba, R. 2009. Network Virtualization: State of the Art and Research Challenges. *IEEE Communications Magazine*, 47(7): 20–26.

Chowdhury, M.; Rahman, M. R.; and Boutaba, R. 2009. Virtual Network Embedding with Coordinated Node and Link Mapping. In *Joint Conference of the IEEE Computer and Communications Societies*, 783–791.

Chowdhury, M.; Rahman, M. R.; and Boutaba, R. 2012. ViNEYard: Virtual Network Embedding Algorithms with Coordinated Node and Link Mapping. *IEEE/ACM Transactions on Networking*, 20(1): 206–219.

Feamster, N.; Gao, L.; and Rexford, J. 2007. How to Lease the Internet in Your Spare Time. *Computer Communication Review*, 37(1): 61–64.

Fischer, A.; Botero, J. F.; Beck, M. T.; de Meer, H.; and Hesselbach, X. 2013. Virtual Network Embedding: A Survey. *IEEE Communications Surveys and Tutorials*, 15(4): 1888–1906.

Gordon, O.; Filmus, Y.; and Salzman, O. 2021. Revisiting the Complexity Analysis of Conflict-Based Search: New Computational Techniques and Improved Bounds. In *International Symposium on Combinatorial Search*, volume 12, 64–72.

Hoenig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *International Conference on Automated Planning and Scheduling*, 477–485.

Ma, H.; Tovey, C.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI Conference on Artificial Intelligence*, 3166–3173.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI Workshop on Planning for Hybrid Systems*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Silver, D. 2005. Cooperative Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 117–122.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Symposium on Combinatorial Search*, 151–159.

Wagner, G. 2015. *Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning*. Ph.D. thesis, Carnegie Mellon University.

Waxman, B. M. 1988. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9): 1617–1622.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.

Yan, Z.; Ge, J.; Wu, Y.; Li, L.; and Li, T. 2020. Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks. *IEEE Journal of Selected Areas in Communications*, 38(6): 1040–1057.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI Conference on Artificial Intelligence*, 1443–1449.

Yu, M.; Yi, Y.; Rexford, J.; and Chiang, M. 2008. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *Computer Communication Review*, 38(2): 17–29.

Zheng, H.; Li, J.; Gong, Y.; Chen, W.; Yu, Z.; Zhan, Z.; and Lin, Y. 2017. Link mapping-oriented ant colony system for virtual network embedding. In *IEEE Congress on Evolutionary Computation*, 1223–1230.

Zhu, Y.; and Ammar, M. H. 2006. Algorithms for Assigning Substrate Network Resources to Virtual Network Components. In *Joint Conference of the IEEE Computer and Communications Societies*, 1–12.