Adapting the Conflict-Based Search Framework for the Virtual Network Embedding Problem

YI ZHENG*, University of Southern California, USA
ERIK KLINE, Information Sciences Institute, University of Southern California, USA
LINCOLN THURLOW, Information Sciences Institute, University of Southern California, USA
SRIVATSAN RAVI, Information Sciences Institute, University of Southern California, USA
SVEN KOENIG, University of California, Irvine, USA
T. K. SATISH KUMAR, University of Southern California, USA

Virtualization is the mechanism of creating virtual representations of physical resources. It is ubiquitously used in data centers and cloud computing services. The objective of virtualization is to ensure that the physical resources are managed efficiently and effectively. This goal induces the Virtual Network Embedding (VNE) problem: the cornerstone task of properly allocating the physical resources of a network to satisfy requests for resources under various constraints while ensuring the quality of service and maximizing resource utilization. Combinatorially, the VNE problem is a problem in resource management that is NP-hard to solve optimally. In this article, we adapt the Conflict-Based Search (CBS) framework for solving the VNE problem, inspired by its success in the Multi-Agent Path Finding (MAPF) domain. We create two algorithms: VNE-CBS and its improved version Improved VNE-CBS (iVNE-CBS). These algorithms not only successfully address the unique challenges in applying the CBS framework to the VNE problem but also import powerful algorithmic techniques, such as disjoint splitting, bypassing conflicts, and conflict avoidance tables, from the MAPF domain. We show that iVNE-CBS significantly outperforms popular baseline VNE algorithms: It scales to networks with several hundred vertices and thousands of edges—significantly larger than the scale of the networks previously used in the VNE literature—while also producing better-quality solutions. The success of our approach might pave the way for overcoming a crucial issue in Internet ossification via heuristic search methods.

JAIR Track: Multi-Agent Path Finding

JAIR Associate Editor: Roni Stern

JAIR Reference Format:
Yi Zheng, Erik Kline, Lincoln Thurlow, Srivatsan Ravi, Sven Koenig, and T. K. Satish Kumar. 2026. Adapting the Conflict-Based Search Framework for the Virtual Network Embedding Problem. *Journal of Artificial Intelligence Research* 86, Article 10 (June 2026), 36 pages. DOI: 10.1613/jair.1.18153

*Corresponding Author.

Authors' Contact Information: Yi Zheng, ORCID: 0009-0001-7890-1787, yzheng63@usc.edu, University of Southern California, Los Angeles, California, USA; Erik Kline, ORCID: 0009-0006-4503-6359, kline@isi.edu, Information Sciences Institute, University of Southern California, Marina del Rey, California, USA; Lincoln Thurlow, ORCID: 0000-0002-8513-4088, lincoln@isi.edu, Information Sciences Institute, University of Southern California, Marina del Rey, California, USA; Srivatsan Ravi, ORCID: 0000-0002-2965-3940, sravi@isi.edu, Information Sciences Institute, University of Southern California, Marina del Rey, California, USA; Sven Koenig, ORCID: 0000-0002-5458-094X, sven.koenig@uci.edu, University of California, Irvine, Irvine, California, USA; T. K. Satish Kumar, ORCID: 0009-0007-8536-3642, tkskwork@gmail.com, University of Southern California, Los Angeles, California, USA.

DOI: 10.1613/jair.1.18153

1 Introduction

It is difficult to make large-scale architectural changes to the Internet. This is referred to as the Internet ossification problem. However, despite the architectural resistance of the Internet, it is amenable to the mobilization of resources as services, often referred to with the "as a service (aaS)" suffix. Examples include Infrastructure aaS (IaaS), Platform aaS (PaaS), Service aaS (SaaS), and Bare-Metal aaS (BMaaS). Network virtualization is a technology that enables resource mobilization on a network, a set of interconnected networks, or the Internet as a whole (Chowdhury and Boutaba 2009). Through virtualization, users obviate the necessity to operate resources directly, reducing the capital and operating costs in a shared environment that has many users and many kinds of resources (Feamster et al. 2007). Internet Service Providers (ISPs) can use virtualization to provide Virtual Network Functions (VNFs) and network slices to their customers, facilitating access to the Internet with various Quality of Service (QoS) guarantees. This capability serves 5G Networks and Network Slice aaS (NSaaS), as described in the 3rd Generation Partnership Project (3GPP) standard (The European Telecommunications Standards Institute 2020).

The physical resources on a network can be of many different types: processors, communication links, and storage devices, among others. While some of these resources are localized to individual network nodes, some other resources, such as communication bandwidths between network nodes, are spread across multiple network links. Network virtualization provides a "logical" representation of these resources, leading to many benefits. However, it introduces an orchestration problem: The physical resources have to be managed efficiently and effectively in a shared environment that may have many users and many kinds of constraints. In essence, the physical resources of a network have to be properly allocated to satisfy requests for resources while meeting various kinds of constraints, ensuring the necessary QoS, and maximizing the overall resource utilization.

This is commonly referred to as the Virtual Network Embedding (VNE) problem. The VNE problem can be understood as satisfying the resource requirements of a Virtual Network Request (VNR) by allocating the physical resources of a Substrate Network (SN) to it (Chowdhury, Rahman, et al. 2009). A VNR is represented by a graph with vertices representing logical compute nodes and edges representing logical communication links between pairs of logical compute nodes. A successful embedding of a VNR onto an SN allocates CPU resources from the physical compute nodes of the SN to each of the VNR vertices and bandwidth resources from the physical communication paths in the SN to each of the VNR edges while satisfying the various CPU and bandwidth capacity constraints.

The VNE problem captures many resource allocation tasks that arise, for example, in computer systems and computer networks. In the context of 5G technology, a customer may request a network slice to support 5G Ultra-Reliable Low Latency Communications (URLLC), specifying the slice ingress to be a certain Radio Access Network (RAN) and the slice egress to be a particular Point of Presence (PoP) (Barakabitze et al. 2020; Popovski et al. 2018; Richart et al. 2016). The task of the network administrator is to efficiently construct the slice across their network, provide QoS guarantees of the requested URLLC, and effectively manage the network resources to maintain the current slice and admit new ones. This task is exemplary of the VNE problem in 5G networks.

The VNE problem is essentially a combinatorial problem that models the proper management of shared resources on a network. It can also be interpreted as a path-coordination problem with constraints (Chowdhury, Rahman, et al. 2009). The VNE problem is NP-hard to solve optimally (M. Yu et al. 2008). In this article, we focus on the core VNE problem with geographical constraints, as proposed in (Chowdhury, Rahman, et al. 2009). Previous works on solving it have developed various kinds of algorithms. However, many of these algorithms either produce low-quality solutions, suffer from scalability issues as the size of the SN or the VNR grows, and/or lack strong theoretical properties.

To address these drawbacks, we present a novel Conflict-Based Search (CBS) framework. Our approach is inspired by the success of the same framework in the Multi-Agent Path Finding (MAPF) domain. A cornerstone

combinatorial problem in this domain is the MAPF problem (Stern et al. 2019). The MAPF problem is characterized by multiple agents in a shared environment that need to coordinate with each other for their pathfinding activities. In this 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 occurs 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. The MAPF problem has many variants and objectives (Stern et al. 2019). For the objective of minimizing the sum of the travel costs of the agents, the MAPF problem is NP-hard to solve optimally (Ma et al. 2016; J. Yu and LaValle 2013).

The CBS framework is the basis of many successful MAPF solvers (Barer et al. 2014; Sharon et al. 2015). It is a two-level search framework, in which the high-level search starts by planning a path for each agent independently. If a conflict occurs 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 as the high-level search proceeds. The low-level search performs single-agent pathfinding to compute optimal individual paths under the spatiotemporal constraints imposed by a high-level CT node.

Combinatorially, both the MAPF problem and the VNE problem can be viewed as constrained path-coordination problems. Therefore, the heuristic search techniques that work well in one problem domain can be transferred to the other. However, the two problems are also different in subtle ways. First, the MAPF problem has a temporal dimension (due to the agent movements) that is absent from the VNE problem. Second, the MAPF problem primarily has mutual exclusion constraints between pairs of agents, whereas the VNE problem primarily has capacity constraints on the CPU and bandwidth resources. Therefore, the transfer of heuristic search techniques requires careful adaptation.

Similar to the CBS framework for solving the MAPF problem, our CBS framework for solving the VNE problem uses a high-level search and a low-level search. The high-level search is a best-first search that resolves conflicts arising from resource contentions. The low-level search is a pathfinding algorithm that allocates resources to each VNR vertex and VNR edge under the constraints imposed by a high-level search node. On the one hand, our approach exploits the similarities between the MAPF problem and the VNE problem. On the other hand, it also successfully addresses the subtle differences between the two problems and the unique challenges that arise in applying the CBS framework to the VNE problem.

Within the CBS framework, we develop two algorithms to solve the VNE problem. The first algorithm, called VNE-CBS, is a vanilla adaptation of the CBS framework to the VNE problem. The second algorithm, called Improved VNE-CBS (iVNE-CBS), draws more power from the CBS literature in the MAPF domain: It incorporates various CBS enhancements whose effectiveness has been recently demonstrated in the MAPF domain. Both algorithms are complete. That is, they find a solution if one exists, and report that no solution exists otherwise. Both algorithms also produce optimal or bounded-suboptimal solutions when requested. A bounded-suboptimal solution is one with a cost that is guaranteed to be within w times of the cost of an optimal solution, for a user-specified value $w \geq 1$. The bounded-suboptimal variants of VNE-CBS and iVNE-CBS have the capability to trade off optimality for increased efficiency.

We highlight several advantages of VNE-CBS and iVNE-CBS over the existing VNE algorithms. On the theoretical front, we prove that both algorithms are complete and can be configured to produce optimal or bounded-suboptimal solutions when required. On the experimental front, we show that iVNE-CBS significantly outperforms other VNE algorithms: It scales to networks with several hundred vertices and thousands of edges—significantly larger than the scale of the networks previously used in the VNE literature—while also producing better-quality solutions. iVNE-CBS outperforms VNE-CBS by virtue of including CBS enhancements, such as disjoint splitting (Li, Harabor, et al. 2019) and bypassing conflicts (Boyarski et al. 2015), in the high-level

search and conflict avoidance tables, true cost heuristics, and more efficient duplicate detection in the low-level search.

iVNE-CBS is practically viable for the scale of the VNE problems that arise in the real world. Moreover, CBS enhancements or other advancements in the MAPF domain can translate to improvements in the VNE domain. Therefore, our approach might pave the way for overcoming crucial issues in Internet ossification via heuristic search methods.

This article is based on two conference papers that we have previously published, namely (Y. Zheng, Ravi, Kline, Koenig, et al. 2022) and (Y. Zheng, Ravi, Kline, Thurlow, et al. 2023). However, this article has a substantial amount of new material in that it presents formal proofs of various theoretical properties of VNE-CBS and iVNE-CBS, which were omitted from the conference papers. Moreover, it presents a unifying perspective on the two algorithms and includes additional experimental results and details for iVNE-CBS that were not present in the conference paper.

2 Background

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

2.1 Virtual Network Embedding

The VNE problem is a constrained resource allocation problem. It defines the task of allocating resources to VNR vertices and edges by mapping them to SN vertices and paths, respectively. The vertex mapping allocates compute resources, such as CPU resources, from the SN vertices to the VNR vertices. The edge mapping allocates communication resources, such as bandwidth resources, from the SN paths to the VNR edges. Collectively, these mappings constitute the VNE mapping, which is 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 more than its available CPU capacity; and (b) for each SN edge, the sum of the bandwidth requirements of all VNR edges that utilize it should be no more than its available bandwidth capacity. Additional constraints may also be imposed on the VNE mapping. For example, each VNR vertex may be constrained to be mapped only to an SN vertex within a certain distance from a preferred geographical location, such as a data center.

The VNE problem can involve the task of embedding one or more VNRs onto an SN. Multiple VNRs can be specified in the context of an offline setting or an online setting. In the offline setting, multiple VNRs are specified simultaneously and can be collectively thought of as a single "composite" VNR. However, this reformulation fails under the commonly used VNE constraint that different VNR vertices from the same VNR have to be mapped to different SN vertices (Chowdhury, Rahman, et al. 2012). In the online setting, multiple VNRs arrive in sequence at different times and stay in the network for a finite period of time.

In the current VNE literature, it has been conventional to empirically evaluate VNE algorithms in both the offline setting and the online setting. In the offline setting, VNE instances with a single VNR are used to evaluate the performance of algorithms. Such VNE instances also impose the constraint of having to map different VNR vertices to different SN vertices. In the online setting, multiple VNRs may hold network resources simultaneously, and, therefore, VNR vertices from different VNRs are allowed to be mapped to the same SN vertex. In this case, reconfiguration may be necessary, that is, it may be necessary to backtrack on the resource allocations made for past VNRs to be able to accommodate newly arrived VNRs.

There are many quality metrics for VNE mappings. They include the revenue, the cost, and the acceptance ratio. The revenue quantifies the total resources demanded by a successfully embedded VNR, while the cost quantifies the total resources meted out to that VNR by the VNE mapping. The revenue depends only on the VNR (provided that it is successfully embedded), but the cost depends on the VNE mapping. In both the offline

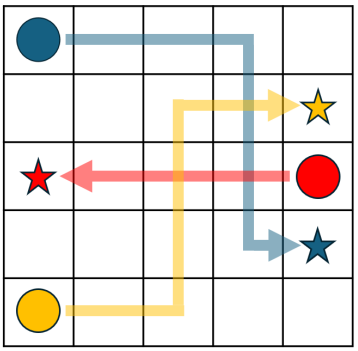

Fig. 1. An example of a MAPF instance where agents (circles) move to goal vertices (stars) in a grid environment.

setting and the online setting, the acceptance ratio quantifies the fraction of successfully embedded VNRs among all VNRs. In both settings, a popular objective is to minimize the cost, while the acceptance ratio is often used as a key quality metric.

The VNE problem and its more expressive variants are known to be NP-hard by virtue of their relationship to the multiway separator problem (M. Yu et al. 2008).

Many algorithms have been proposed for solving the VNE problem and its variants. A compendium of existing algorithms can be found in several survey articles (Cao et al. 2019; Fischer et al. 2013). VNE algorithms that use mathematical optimization usually model the VNE problem with Mixed Integer Programming (MIP) or Integer Linear Programming (ILP). For example, the D-ViNE and R-ViNE algorithms use deterministic and randomized rounding techniques, respectively, to extract a solution from a Linear Programming (LP) relaxation of an MIP model (Chowdhury, Rahman, et al. 2009).

VNE algorithms that use node ranking map VNR vertices to SN vertices greedily according to various heuristics. Subsequently, the VNR edges are mapped to SN paths using regular shortest path or multi-commodity flow computations: G-SP and G-MCF are two such algorithms that use shortest path and multi-commodity flow computations, respectively (M. Yu et al. 2008; Zhu and H. Ammar 2006). Other VNE algorithms, such as RW-MaxMatch-SP, are inspired by Google's Page Rank algorithm (Cheng et al. 2011). These algorithms use the Markov Random Walk model to rank the VNR and the SN vertices based on their demanded resources and available resources, respectively, as well as their topological attributes. They subsequently utilize these ranks to map the VNR vertices to the SN vertices.

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

2.2 Multi-Agent Path Finding

The MAPF problem is specified by an undirected unweighted graph $G = (V, E)$ and a set of k agents $\{a_1, a_2 \dots a_k\}$, where a_i is required to move from a start vertex $s_i \in V$ to a goal vertex $g_i \in V$. Time is discretized into timesteps. At each timestep, each agent can either move to an adjacent vertex or wait at its current vertex, both with unit cost. A path of a_i is a sequence of move and wait actions that lead a_i from s_i to g_i. A vertex conflict (a_i, a_j, v, t) occurs when two different agents a_i and a_j are at the same vertex v at the same timestep t. An edge conflict (a_i, a_j, u, v, t) occurs when two different agents a_i and a_j traverse the same edge (u, v) in opposite directions between the same timesteps t and $t+1$. A solution of a MAPF instance is a set of paths, one for each agent, without

conflicts. One common objective is to find a solution that minimizes the sum of the costs incurred by all agents along their paths (Sharon et al. 2015). Solving the MAPF problem optimally for this objective is NP-hard (Ma et al. 2016; J. Yu and LaValle 2013). Figure 1 shows a MAPF instance in a grid environment. Several variants of the MAPF problem—that are also NP-hard—have been proposed to model different real-world scenarios (Stern et al. 2019). The MAPF problem and its variants have many real-world applications, including in video games (Silver 2005), automated warehousing (R. Wurman et al. 2008), traffic management (M. Dresner and Stone 2008), and multi-drone delivery (Choudhury et al. 2020).

2.3 Conflict-Based Search

CBS is a two-level heuristic search framework that has been developed for solving the MAPF problem optimally (Sharon et al. 2015). On the high level, CBS performs a best-first search on a CT. Each CT node N contains a set of spatiotemporal constraints $N.constraints$ that are used to avoid conflicts among the agents. CT node N also contains a set of paths $N.paths$, one for each agent, that respects $N.constraints$. The cost of CT node N is the sum of the costs of the paths in $N.paths$. The root CT node contains an empty set of constraints and a set of shortest paths that may contain conflicts. When CBS expands a CT node N, it first checks for conflicts between every pair of paths in $N.paths$. If there are none, the CT node is a goal CT node and CBS returns its set of paths as the solution. Otherwise, CBS chooses one of the conflicts and resolves it by splitting N into two child CT nodes, each with an additional constraint prohibiting one agent involved in the conflict from using the conflicting vertex or conflicting edge at the conflicting timestep. CBS then uses its low-level search, such as A^*, to replan the path of this agent so that it satisfies the new constraint. The fewer conflicts there are to resolve, the faster CBS terminates. A complexity analysis of CBS is presented in (Gordon et al. 2021). CBS guarantees returning a solution if one exists because it explores both ways of resolving each conflict. Moreover, it guarantees optimality of the returned solution because it performs best-first search on both its high and low levels.

Since solving the MAPF problem optimally is hard, suboptimal solution procedures can be investigated to increase the runtime efficiency. Enhanced CBS (ECBS) has been developed to produce suboptimal solutions in the CBS framework by trading off the solution cost for runtime (Barer et al. 2014). ECBS utilizes the power of a bounded-suboptimal search algorithm called focal search. Focal search maintains two lists of nodes *OPEN* and *FOCAL*. *OPEN* is the regular open list, as in A^*, whose nodes n are sorted by a cost function $f(n) = g(n) + h(n)$, where $h(n)$ is an admissible heuristic function, that is, one that never overestimates the true cost. Let $w \geq 1$ be a user-specified suboptimality factor and $f_{min} = \min_{n_i \in OPEN} f(n_i)$ be the minimum f-value of any node in *OPEN*. *FOCAL* contains the nodes n in *OPEN* for which $f(n) \leq w \cdot f_{min}$, sorted by a secondary heuristic function $d(n)$ that estimates the smallest number of hops from node n to a goal node. $d(n)$ can be inadmissible. Unlike A^*, focal search always expands a node n with the minimum d-value in *FOCAL*. Let C^* be the cost of an optimal solution. Focal search guarantees that the cost of the returned solution is at most $w \cdot C^*$ since f_{min} is a lower bound on C^*. ECBS is a bounded-suboptimal variant of CBS whose high-level and low-level searches are both focal searches. Both searches use measures related to the number of conflicts as the secondary heuristic function $d(n)$. ECBS(w) refers to ECBS with the user-specified suboptimality factor w to be used in its focal searches. ECBS(w) is w-suboptimal, that is, it always returns a solution—if one exists—with a cost that is no more than $w \cdot C^*$ (Barer et al. 2014; Cohen et al. 2016). ECBS(w), for a reasonably small value of w, has the flexibility of expanding CT nodes with fewer conflicts than the CT nodes chosen for expansion by CBS. This often makes ECBS(w) faster than CBS.

3 Problem Formulation

In this article, we focus on the core VNE problem with geographical constraints, as proposed in (Chowdhury, Rahman, et al. 2009). In this section, we describe the problem.

3.1 Substrate Network

We define an SN as an undirected graph $G^s = (V^s, E^s, A_V^s, A_E^s)$, where 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 $\textsc{cpu}(v^s)$ and its geographical location $\textsc{loc}(v^s)$. The attribute of an SN edge $e^s \in E^s$ is its bandwidth capacity $\textsc{bw}(e^s)$. An SN path is a path in G^s.

3.2 Virtual Network Request

We define a VNR as an undirected graph $G^r = (V^r, E^r, C_V^r, C_E^r)$, where V^r is the set of VNR vertices, E^r is the set of VNR edges, C_V^r is a mapping from VNR vertices to their demands, and C_E^r is a mapping from VNR edges to their demands. The demands of a VNR vertex $v^r \in V^r$ are its CPU requirement $\textsc{cpu}(v^r)$, its preferred geographical location $\textsc{loc}(v^r)$, and the maximum allowed distance $D(v^r)$ from its preferred geographical location to the location of the SN vertex that it is mapped to. The popularly considered demand of a VNR edge $e^r \in E^r$ is its bandwidth requirement $\textsc{bw}(e^r)$.

3.3 Virtual Network Embedding

As mentioned before, it has been conventional to evaluate VNE algorithms on single VNRs in the offline setting. Hence, we will formalize the VNE problem as the problem of finding a VNE mapping for a single VNR. The same formulation suffices in the online setting: Although multiple VNRs can be simultaneously embedded onto an SN, they are processed one at a time, each time considering only the unallocated SN resources.[1]

Given a VNR G^r, a feasible VNE mapping $\textsc{vne}(\cdot)$ maps VNR vertices to SN vertices and VNR edges to SN paths so as to satisfy the following constraints.

For the vertex mapping of VNR vertices to SN vertices,

(1) each VNR vertex $v^r \in V^r$ is mapped to exactly one SN vertex $\textsc{vne}(v^r) \in V^s$,
(2) no two VNR vertices $v_i^r \in V^r$ and $v_j^r \in V^r$ are mapped to the same SN vertex $v^s \in V^s$, and
(3) for each VNR vertex $v^r \in V^r$:

$$\textsc{cpu}(v^r) \leq \textsc{cpu}(\textsc{vne}(v^r))$$

and

$$\textsc{geodist}(\textsc{loc}(v^r), \textsc{loc}(\textsc{vne}(v^r))) \leq D(v^r),$$

where $\textsc{geodist}(\cdot, \cdot)$ is a function that calculates the distance between two geographical locations.

For the edge mapping of VNR edges to SN paths,

(1) each VNR edge $(v_i^r, v_j^r) \in E^r$ is mapped to an SN path $\textsc{vne}((v_i^r, v_j^r))$ from $\textsc{vne}(v_i^r)$ to $\textsc{vne}(v_j^r)$ in G^s, and
(2) for each SN edge $e^s \in E^s$, the sum of the bandwidth requirements of all VNR edges that utilize it should be no more than its available bandwidth capacity:

$$\sum\nolimits_{e^r \in E^r:\ e^s \in \textsc{vne}(e^r)} \textsc{bw}(e^r) \leq \textsc{bw}(e^s).$$

3.4 Objectives

Several metrics are commonly used for evaluating VNE mappings (Fischer et al. 2013). One of them is the revenue. It is the sum of the resources requested by the successfully embedded VNR. It is 0 if the VNR cannot be embedded

[1] If multiple VNRs are considered simultaneously, they can be treated as a single "composite" VNR. However, as mentioned before, this reformulation fails if different VNR vertices from the same VNR have to be mapped to different SN vertices. In such a case, the VNE-CBS algorithms themselves can be adapted accordingly.

onto the SN. Otherwise, it is given by the expression:

$$\sum_{v^r \in V^r} \text{CPU}(v^r) + \sum_{e^r \in E^r} \text{BW}(e^r). \tag{1}$$

Another metric is the cost. It is the sum of the SN resources that are utilized for embedding the VNR. It is 0 if the VNR cannot be embedded onto the SN. Otherwise, it is given by the expression:

$$\sum_{v^r \in V^r} \text{CPU}(v^r) + \sum_{e^r \in E^r} \sum_{e^s \in \text{VNE}(e^r)} \text{BW}(e^r). \tag{2}$$

The cost of embedding a VNR is larger than or equal to its revenue since every VNR edge utilizes an SN path with at least one SN edge in it.[2] Another popular metric is the cost/revenue ratio representing the cost divided by the revenue. It measures how efficiently the SN resources are allocated to a VNR.

In the offline setting, multiple VNRs are embedded onto an SN one at a time without considering other VNRs. In the online setting, multiple VNRs arrive in sequence at different times and stay in the network for a finite period of time: The SN resources are held by previously embedded VNRs that have not yet left. In both the offline and the online settings, the objective is to maximize profit, that is, revenue minus cost. However, since unit prices are often dropped for convenience, a popular placeholder objective in both settings is to minimize the cost. Of course, VNE algorithms with this primary objective can also be evaluated on other quality metrics, such as the acceptance ratio and the cost/revenue ratio.

3.5 Reformulating the VNE Problem as a Path-Coordination Problem

The VNE problem can be interpreted as a path-coordination problem after some reformulations (Chowdhury, Rahman, et al. 2009).

Given a VNR G^r and an SN G^s, we construct an augmented graph G^m that combines the SN and the VNR. For each VNR vertex $v^r \in V^r$, we introduce exactly one fictitious vertex $v^f \in V^f$ to represent v^r in G^m. Each fictitious vertex v^f inherits all attributes of the corresponding VNR vertex v^r, such as $\text{CPU}(v^r)$, $\text{LOC}(v^r)$, and $D(v^r)$. Each fictitious vertex v^f is connected via fictitious undirected edges $(v^f, v^s) \in E^f$ to all SN vertices $v^s \in V^s$ that satisfy the CPU constraint $\text{CPU}(v^f) \leq \text{CPU}(v^s)$ and the geographical constraint $\text{GEODIST}(\text{LOC}(v^f), \text{LOC}(v^s)) \leq D(v^f)$. Each fictitious edge has infinite bandwidth capacity and represents the vertex mapping of a VNR vertex to an SN vertex. Together, the fictitious vertices and edges extend the SN, completing the construction of the augmented graph G^m, where the vertices are $V^s \cup V^f$ and the edges are $E^s \cup E^f$.

A VNE mapping is a set of paths in G^m. Each VNR edge (v_i^r, v_j^r) that is mapped to an SN path can be traced in G^m with starting and ending fictitious edges (v_i^f, v_1^s) and (v_2^s, v_j^f), respectively, where $v_1^s, v_2^s \in V^s$. The mapping of VNR vertices v_i^r and v_j^r to SN vertices v_1^s and v_2^s, respectively, can be identified by the utilization of the fictitious edges (v_i^f, v_1^s) and (v_2^s, v_j^f) in the path, where v_i^f and v_j^f are the fictitious vertices corresponding to v_i^r and v_j^r, respectively. Such a path in G^m cannot have any other fictitious vertices or edges. The VNE problem can then be interpreted as a path-coordination problem with CPU and bandwidth constraints. Section 3.6 shows an example.

In general, for any feasible VNE mapping of a given VNR onto the SN, Equation 2 indicates that the cost of the VNE mapping depends on the allocated CPU and bandwidth resources. The CPU resources allocated to the VNR vertices are identical for all feasible VNE mappings. However, the bandwidth resources allocated to the VNR edges depend on how the VNR edges are mapped to the SN paths. In particular, the cost of mapping each VNR edge $e^r \in E^r$ is its bandwidth requirement $\text{BW}(e^r)$ multiplied by the length of the SN path that it is mapped to.

[2] In the real world, the ISP obtains the revenue from the customers and incurs the cost from the infrastructure providers. The unit price that the ISP charges the customers is higher than the unit price that the ISP pays the infrastructure providers. Hence, the real-world revenue is higher than the real-world cost for the ISP. However, it is convenient to use the definitions of the revenue and the cost without the unit prices for the study of VNE algorithms.

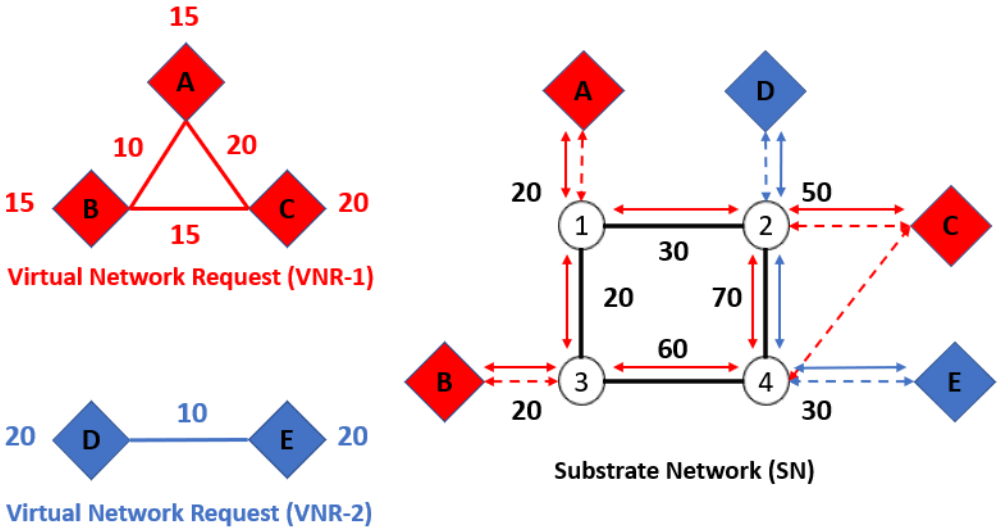

Fig. 2. An example embedding of two VNRs, VNR-1 and VNR-2, onto an SN.

Thus, minimizing the cost of a VNE mapping is the same as minimizing the sum of the lengths of the chosen paths in G^m. In turn, this corresponds to the objective of minimizing the "sum of costs" in the CBS framework.

3.6 Example

Figure 2 shows an example embedding of two VNRs onto an SN by reformulating the VNE problem as a path-coordination problem. On the left side, it shows the two VNRs, VNR-1 (in red) and VNR-2 (in blue). On the right side, it shows the SN (in black) with additional fictitious vertices and edges, as described for the construction of G^m. The non-negative numbers annotating the VNR vertices and the SN vertices represent their CPU requirements and CPU capacities, respectively. The non-negative numbers annotating the VNR edges and the SN edges represent their bandwidth requirements and bandwidth capacities, respectively. As described earlier, each fictitious vertex represents a VNR vertex and is connected via fictitious edges to the SN vertices that it can be mapped to. This mechanism represents the geographical constraints. For example, in Figure 2, the VNR vertex C can be mapped to either SN vertex 2 or SN vertex 4 according to its geographical constraints. VNR-1 has the vertex mapping A–1, B–3, and C–2, and its VNR edges A–B, A–C, and B–C are mapped to the SN paths 1–3, 1–2, and 3–4–2, respectively. VNR-2 has the vertex mapping D–2 and E–4, and its VNR edge D–E is mapped to the SN path 2–4.

4 VNE-CBS

In this section, we describe how to adapt the CBS framework to the VNE problem. While doing so, we also describe our VNE-CBS algorithm step by step. Intuitively, the VNE-CBS algorithm uses a high-level search and a low-level search. The high-level search is a best-first search on a CT that resolves conflicts caused by resource contentions. The low-level search is a pathfinding algorithm on the augmented graph that allocates resources to each VNR vertex and VNR edge under the constraints imposed by a high-level search node.

Compared to the CBS algorithm for solving the MAPF problem, our VNE-CBS algorithm adapts both the conflict definitions and the CT constraints to capture the unique aspects of the VNE problem. It adapts the high-level search to resolve VNE-specific conflicts via branching. It also adapts the low-level search to handle

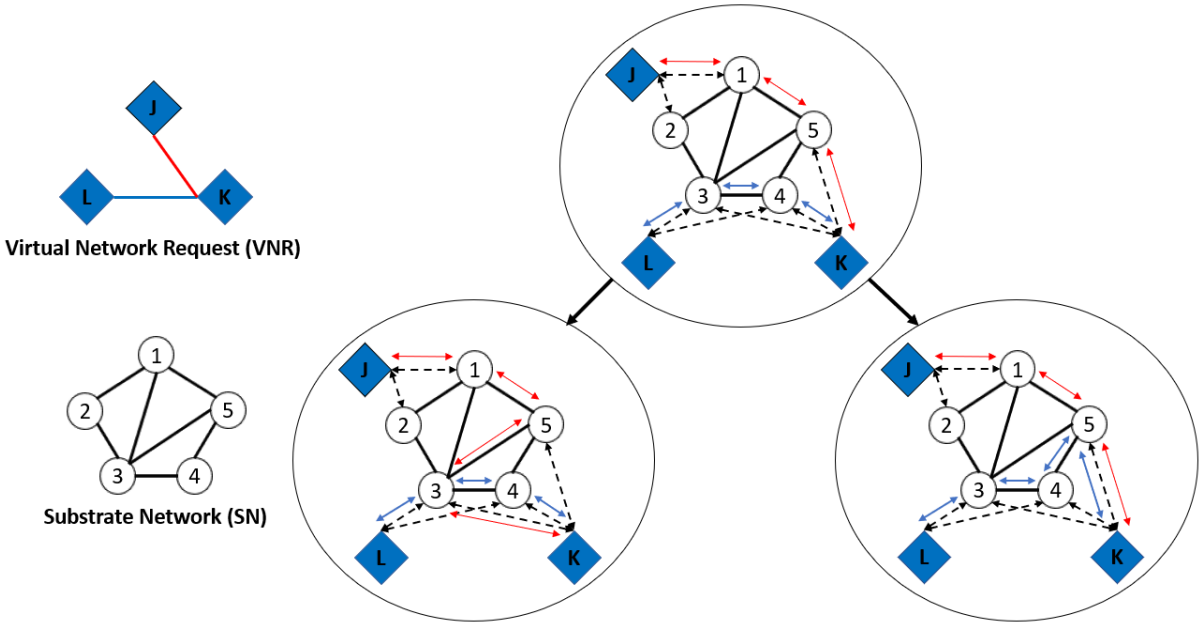

Fig. 3. An example of resolving a type-1 vertex conflict in VNE-CBS.

VNE-specific requirements, including the simultaneous mapping of VNR vertices to SN vertices and VNR edges to SN paths. In Section 4.1, we describe the conflicts and CT constraints that guide the search of VNE-CBS. In Sections 4.2 and 4.3, we describe the high-level and low-level searches of VNE-CBS, respectively.

4.1 Conflicts and CT Constraints

The VNE problem, formalized in Section 3, requires us to satisfy several kinds of constraints. We refer to violations of these constraints as conflicts. The high-level search of VNE-CBS is a conflict-resolution procedure. It starts with a tentative VNE mapping and resolves conflicts in this mapping by using a best-first search. The resolution of a conflict is done via branching, that is, injecting CT constraints into each of the child CT nodes. Hence, each CT node N in the high-level search maintains a set of CT constraints $N.constraints$ that have to be respected in the low-level search.

A type-1 vertex conflict (v^r, v_1^s, v^r, v_2^s) arises when a VNR vertex v^r is mapped to two different SN vertices v_1^s and v_2^s.[3] To resolve it, two child CT nodes are created in the high-level search. One of them enforces the negative constraint $\overline{(v^r, v_1^s)}$ that stops v^r from being mapped to v_1^s, and the other one enforces the negative constraint $\overline{(v^r, v_2^s)}$ that stops v^r from being mapped to v_2^s.

Figure 3 presents an example of resolving a type-1 vertex conflict. It shows a VNR and example CT nodes for embedding it onto the accompanying SN. The root CT node contains a VNE mapping with a type-1 vertex conflict $(K, 4, K, 5)$ since the fictitious vertex K is mapped to two SN vertices 4 and 5. To resolve this vertex conflict, we create two child CT nodes in the high-level search. The left child CT node has a new negative constraint $\overline{(K, 5)}$ that stops the fictitious vertex K from being mapped to the SN vertex 5; and the red path is rerouted to

[3]The first and third arguments in (v^r, v_1^s, v^r, v_2^s) are identical but are retained for convenience.

accommodate the new constraint. Similarly, the right child CT node has a new negative constraint $\overline{(K, 4)}$ that stops the fictitious vertex K from being mapped to the SN vertex 4; and the blue path is rerouted. The left child CT node still has a type-1 vertex conflict since K is mapped to two SN vertices 3 and 4; and this vertex conflict can be resolved by expanding the left child CT node in the same manner. The right child CT node has no vertex conflicts but may be expanded further if there are other conflicts.

A type-2 vertex conflict (v_1^r, v^s, v_2^r, v^s) arises when two VNR vertices v_1^r and v_2^r from the same VNR are mapped to the same SN vertex v^s. To resolve this vertex conflict, we create two child CT nodes in the high-level search. One of them has a new negative constraint $\overline{(v_1^r, v^s)}$ that stops v_1^r from being mapped to v^s, and the other one has a new negative constraint $\overline{(v_2^r, v^s)}$ that stops v_2^r from being mapped to v^s.

A bandwidth capacity conflict arises when multiple VNR edges utilize an SN edge e^s that does not have sufficient bandwidth capacity to accommodate all their bandwidth requirements. To resolve this conflict, 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 e^r with a new negative constraint that stops e^r from utilizing e^s.

The CPU constraints and the geographical constraints of a VNE instance are already enforced in the augmented graph G^m by construction. The high-level search therefore needs to resolve only the type-1 vertex conflicts, the type-2 vertex conflicts, and the bandwidth capacity conflicts.

4.2 High-Level Search

The VNE-CBS algorithm uses a high-level focal search with a user-specified suboptimality factor $w \geq 1$. Algorithm 1 shows the pseudocode of its high-level search. It 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, VNE-CBS uses G^s and G^r to create an augmented graph G^m containing the fictitious vertices and edges, as described in Section 3.5. On Lines 3–7, it initializes the root CT node N_R. On Line 5, it uses the low-level search to find a path from v_1^f to v_2^f in G^m for each VNR edge (v_1^r, v_2^r), where v_1^f and v_2^f are the fictitious vertices corresponding to v_1^r and v_2^r, respectively. $N_R.mapping$ is the resulting set of paths, that is, the initial VNE mapping. $N_R.cost$ is the cost of the mapping, and $N_R.num\_conf$ is the number of conflicts in the mapping. On Line 8, VNE-CBS inserts N_R into priority queues *OPEN* and *FOCAL*. 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*. *FOCAL* maintains a list of CT nodes N in *OPEN* for which $N.cost \leq w \cdot OPEN.top().cost$. 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*.

On Lines 9–30, VNE-CBS expands CT nodes until either a feasible VNE mapping is found or *FOCAL* is empty. On Lines 11–12, it retrieves the node N_T 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 $N_T.mapping$. Otherwise, on Lines 15–16, it finds a conflict *Conf* in $N_T.mapping$ and generates negative constraints *Cons* to resolve it, as described in Section 4.1. It first checks the mapping of each VNR vertex and then checks the mapping of each VNR edge. In case multiple conflicts exist, it prefers to resolve vertex conflicts before bandwidth capacity conflicts. This prioritization prevents the algorithm from spending unnecessary effort on mapping VNR edges to SN paths when the start and end edges of these paths—that represent the vertex mappings—are invalid. Beyond this preference, it resolves conflicts in the order in which they are found in $N_T.mapping$.

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 $N_C.constraints$. If C already exists in $N_C.constraints$, VNE-CBS does not create a new child CT node. On Line 20, it updates $N_C.mapping$ to accommodate the added constraint by invoking the low-level search to recompute the paths that do not respect the added constraint. If the path update

Algorithm 1 High-Level Search of VNE-CBS.

1: Input: G^s, G^r, w
2: $G^m \leftarrow$ augmented graph for G^s and G^r
3: $N_R \leftarrow$ *empty CT node*
4: $N_R.constraints \leftarrow \emptyset$
5: $N_R.mapping \leftarrow$ low-level paths found in G^m for all VNR edges using the low-level search
6: $N_R.cost \leftarrow \texttt{cost}(N_R.mapping)$
7: $N_R.num\_conf \leftarrow$ number of conflicts in $N_R.mapping$
8: $OPEN = FOCAL = \{N_R\}$
9: **while** $FOCAL \neq \emptyset$ **do**
10: $cost_{old} \leftarrow OPEN.top().cost$
11: $N_T \leftarrow FOCAL.top()$
12: Remove N_T from *OPEN* and *FOCAL*
13: **if** $N_T.num\_conf = 0$ **then**
14: **return** $N_T.mapping$ as the solution
15: $Conf \leftarrow$ first conflict found in $N_T.mapping$
16: $Cons \leftarrow$ constraints for resolving $Conf$
17: **for** $C \in Cons$ **do**
18: $N_C \leftarrow$ copy of N_T
19: Add C to $N_C.constraints$
20: Update the low-level paths in $N_C.mapping$ to accommodate the constraint C
21: **if** update is successful **then**
22: $N_C.cost \leftarrow \texttt{cost}(N_C.mapping)$
23: $N_C.num\_conf \leftarrow$ number of conflicts in $N_C.mapping$
24: $OPEN \leftarrow OPEN \cup \{N_C\}$
25: **if** $N_C.cost \leq w \cdot cost_{old}$ **then**
26: $FOCAL \leftarrow FOCAL \cup \{N_C\}$
27: $cost_{new} \leftarrow OPEN.top().cost$
28: **for** $N \in OPEN$ **do**
29: **if** $w \cdot cost_{old} < N.cost \leq w \cdot cost_{new}$ **then**
30: $FOCAL \leftarrow FOCAL \cup \{N\}$
31: **return** "No Solution"

on Lines 20–21 succeeds, VNE-CBS calculates the cost and the number of conflicts for the child CT node N_C on Lines 22–23 and inserts N_C into *OPEN* on Line 24. On Lines 25–26, following the standard focal search procedure, it inserts N_C into *FOCAL* if $N_C.cost$ is within w times the cost of the top CT node in *OPEN* from Line 10. On Lines 27–30, VNE-CBS updates *FOCAL* in case the cost of the top CT node in *OPEN* has increased as a result of the previous operations. On Line 31, VNE-CBS reports the absence of a solution.

4.3 Low-Level Search

Algorithm 2 shows the pseudocode of the low-level search of VNE-CBS. N is the CT node from the high-level search that sets up the context of the low-level search. Algorithm 2 finds a path from a fictitious vertex v_1^f to another fictitious vertex v_2^f in G^m while respecting the constraints in $N.constraints$. The fictitious vertices v_1^f and v_2^f represent the given VNR vertices v_1^r and v_2^r, respectively. As mentioned before, each fictitious vertex $v^f \in V^f$

Algorithm 2 Low-Level Search of VNE-CBS.

```
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, respectively
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)
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
```

inherits all attributes of the corresponding VNR vertex v^r, resulting in $\text{CPU}(v^f) = \text{CPU}(v^r)$, $\text{LOC}(v^f) = \text{LOC}(v^r)$, and $D(v^f) = D(v^r)$.

Algorithm 2 takes five inputs: the two VNR vertices v_1^r and v_2^r, the augmented graph G^m, and the two attributes of the high-level CT node N: $N.constraints$ and $N.mapping$. It outputs a path from v_1^f to v_2^f of minimum length that respects $N.constraints$. On Lines 2–6, Algorithm 2 performs initializations and sets up the appropriate data structures for carrying out the subsequent search procedure on Lines 7–16. On Lines 8–9, the algorithm retrieves the search node n_T to be expanded next. On Lines 10–11, n_T is recognized as a goal node if $n_T = v_2^f$ and the grandparent[4] of n_T is not v_1^f. This condition on the goal node prevents the case where a path of length 2, that is, $\langle v_1^f, v^s, v_2^f \rangle$, gets spuriously returned as a solution path. Such a path is not a solution path since it maps the two VNR vertices v_1^r and v_2^r to the same SN vertex v^s, resulting in a type-2 vertex conflict.

On Line 12, the function `valid_neighbors`$(\cdot)$ returns a list of valid neighbors of n_T. When $n_T = v_1^f$, it includes all SN vertices v^s in the return list that satisfy $\text{CPU}(v^s) \geq \text{CPU}(n_T)$ and $\text{GEODIST}(\text{LOC}(n_T), \text{LOC}(v^s)) \leq D(n_T)$. When n_T is an SN vertex, it (a) includes all SN vertices v^s in the return list that satisfy $\text{BW}(e^s) \geq \text{BW}(e^r)$, where $e^s = (n_T, v^s)$ and $e^r = (v_1^r, v_2^r)$, (b) includes v_2^f in the return list if v_2^f is one of the neighbors of n_T and satisfies $\text{CPU}(n_T) \geq \text{CPU}(v_2^f)$ and $\text{GEODIST}(\text{LOC}(v_2^f), \text{LOC}(n_T)) \leq D(v_2^f)$, and (c) excludes fictitious vertices that are not v_2^f, since a path is not allowed to contain any fictitious vertices other than v_1^f and v_2^f. In all cases, it excludes the neighboring vertices that are ruled out by the negative constraints in $N.constraints$. On Lines 13–15, Algorithm 2 generates a child node for each valid neighbor n, assigns its g-value to be 1 larger than that of n_T, and puts it into $OPEN$. On Line 16, n_T is added to $CLOSED$. On Line 17, the algorithm returns failure if no path can be found from v_1^f to v_2^f.

On Line 8, Algorithm 2 uses a tie-breaking rule for selecting a search node from $OPEN$ among all those search nodes with the minimum g-value. If two or more SN vertices have the same g-value, tie-breaking chooses an

[4]The grandparent is the parent of the parent, where the parent relationship is indicated by `par`$(\cdot)$.

SN vertex that is assigned the fewest VNR vertices. This tie-breaking rule keeps the number of type-2 vertex conflicts small and thus has the potential to reduce the number of CT node expansions for finding a feasible VNE mapping in the high-level search.

Unlike the low-level search of ECBS in the MAPF domain, the low-level search of VNE-CBS does not use a focal search. The absence of a temporal dimension in the VNE problem generally results in short paths, making a focal search unnecessary.

4.4 Theoretical Properties of VNE-CBS

We now prove some properties of VNE-CBS, analogous to the properties of the CBS algorithm established in the MAPF domain (Sharon et al. 2015). From Lines 13–14 of Algorithm 1, it is easy to observe that VNE-CBS is sound, that is, it never returns an infeasible solution. In Section 4.4.1, we first prove its optimality for $w = 1$ and then prove its w-suboptimality for any value of $w > 1$. In Section 4.4.2, we prove its completeness, that is, VNE-CBS always returns a solution if one exists and correctly reports that a VNE instance is unsolvable if no solution exists. As expected, these proofs are analogous to the proofs presented in (Sharon et al. 2015).

4.4.1 w-Suboptimality. Consider Algorithm 1. For a given CT node N, let $CV(N)$ be the set of all feasible VNE mappings that also satisfy $N.constraints$. We say that the CT node N permits a feasible VNE mapping p if $p \in CV(N)$. $CV(N_R)$ is the set of all feasible VNE mappings since $N_R.constraints = \emptyset$. If N is not a goal CT node, $N.mapping \notin CV(N)$ because $N.mapping$ is not feasible. Let $minCost(CV(N))$ be the minimum cost over all feasible VNE mappings in $CV(N)$. If $CV(N) = \emptyset$, we define $minCost(CV(N))$ to be ∞.

Lemma 4.1. *For a CT node N, $N.cost \leq minCost(CV(N))$.*

Proof. The cost of mapping the VNR vertices to the SN vertices is the same for all feasible VNE mappings; but the cost of mapping the VNR edges to the SN paths can be different. The low-level search of VNE-CBS finds a shortest SN path that satisfies $N.constraints$ to implement each VNR edge independent of the implementation of the other VNR edges. Hence, $N.mapping$ is the optimal cost mapping that satisfies $N.constraints$. Since the set of feasible VNE mappings that satisfy $N.constraints$ is a subset of the set of VNE mappings that satisfy $N.constraints$, $N.cost \leq minCost(CV(N))$. □

Lemma 4.2. *For each feasible VNE mapping p and at all stages of Algorithm 1, there exists at least one CT node N in OPEN that permits p.*

Proof. We prove this by induction. In the base case, VNE-CBS initializes *OPEN* with the root CT node N_R. Since $N_R.constraints = \emptyset$, N_R permits all feasible VNE mappings. In the inductive case, assume that the claim holds after the first $i - 1$ CT node expansions. We now prove that it then also holds after the i-th CT node expansion. Let N be the i-th expanded CT node. If p is a feasible VNE mapping that is permitted by any CT node $N' \neq N$ in *OPEN* directly prior to N's expansion, it continues to be permitted by the same CT node N' after N's expansion. However, if p is permitted only by N prior to N's expansion, we show that it is permitted by at least one of N's child CT nodes after N's expansion. We note that N is expanded only in recognition of a conflict in $N.mapping$. This conflict can be of three possible kinds.

- If the conflict is a type-1 vertex conflict (v^r, v_1^s, v^r, v_2^s), VNE-CBS generates two child CT nodes N_1 and N_2 with N_1 recording the constraint $\overline{(v^r, v_1^s)}$ and N_2 recording the constraint $\overline{(v^r, v_2^s)}$. For proof by contradiction, assume that p is not permitted by both N_1 and N_2. Since p is permitted by N but not by N_1, it must violate the constraint $\overline{(v^r, v_1^s)}$, that is, p maps v^r to v_1^s. Similarly, p also maps v^r to v_2^s, contradicting the fact that p is a feasible VNE mapping.
- If the conflict is a type-2 vertex conflict (v_1^r, v^s, v_2^r, v^s), VNE-CBS generates two child CT nodes N_1 and N_2 with N_1 recording the constraint $\overline{(v_1^r, v^s)}$ and N_2 recording the constraint $\overline{(v_2^r, v^s)}$. For proof by contradiction,

assume that p is not permitted by both N_1 and N_2. Since p is permitted by N but not by N_1, it must violate the constraint $\overline{(v_1^r, v^s)}$, that is, p maps v_1^r to v^s. Similarly, p also maps v_2^r to v^s, contradicting the fact that p is a feasible VNE mapping.

- If the conflict is a bandwidth capacity conflict pertaining to the SN edge e^s and can be expressed in the form $\sum_{e^r \in E^r:\ e^s \in \text{VNE}(e^r)} \text{BW}(e^r) > \text{BW}(e^s)$, VNE-CBS generates $|E_c|$ different child CT nodes, where $E_c = \{e^r \in E^r :\ e^s \in \text{VNE}(e^r)\}$. For each VNR edge $e_c^r \in E_c$, VNE-CBS generates a child CT node N_c with N_c recording the constraint that stops e_c^r from utilizing e^s. For proof by contradiction, assume that p is not permitted by all the child CT nodes. Since p is permitted by N but not by any of the child CT nodes N_c, it must violate the constraints recorded by all N_c, that is, it implements every $e_c^r \in E_c$ by utilizing e^s, contradicting the fact that p is a feasible VNE mapping.

□

Theorem 4.3. *For $w = 1$, a solution returned by VNE-CBS is optimal.*

Proof. Consider any feasible VNE mapping p. Let its cost be $p.cost$. From Lemma 4.2, we know that all feasible VNE mappings are permitted at all stages by at least one CT node in *OPEN*. Let $N(p)$ be a CT node in *OPEN* that permits p at the time that a goal CT node N_G is chosen for expansion. When this happens, VNE-CBS returns a solution of cost $N_G.cost$. Since VNE-CBS expands CT nodes in a best-first manner, $N_G.cost \leq N(p).cost$. Moreover, from Lemma 4.1, $N(p).cost$ is no more than $minCost(CV(N(p)))$, which in turn is no more than $p.cost$, since $p \in CV(N(p))$. Hence, $N_G.cost \leq N(p).cost \leq p.cost$, that is, the cost of the solution returned by VNE-CBS is no more than the cost of any other feasible VNE mapping. □

Theorem 4.4. *For any $w > 1$, a solution returned by VNE-CBS is w-suboptimal.*

Proof. Similar to the invocation of Lemma 4.1 in the proof of Theorem 4.3, $N(p).cost \leq p.cost$. $N_G.cost \leq w \cdot \tilde{N}.cost$, where $\tilde{N}$ is a CT node of minimum cost in *OPEN* at the time that N_G is chosen for expansion by the focal search procedure of VNE-CBS. Since $\tilde{N}$ has the minimum cost, $\tilde{N}.cost \leq N(p).cost$. Combining the above inequalities, we have $N_G.cost \leq w \cdot \tilde{N}.cost \leq w \cdot N(p).cost \leq w \cdot p.cost$. Therefore, the cost of the solution returned by VNE-CBS is no more than w times the cost of any other feasible VNE mapping. □

4.4.2 Completeness. In this subsection, we show that VNE-CBS is a complete algorithm, irrespective of the value of $w \geq 1$. Our proof of completeness is simpler than the proof for CBS in the MAPF domain. This is primarily due to the absence of a temporal dimension in the VNE problem.

Lemma 4.5. *For $w \geq 1$, the CT of VNE-CBS has a finite size.*

Proof. We prove the lemma by showing that the CT has a finite branching factor and a finite depth. The branching factor of the CT is upper bounded by the maximum number of child CT nodes that VNE-CBS generates for any CT node while resolving a chosen conflict. While resolving vertex conflicts results in a branching factor of 2, resolving bandwidth capacity conflicts results in a branching factor of $|E_c|$ (as defined in the proof of Lemma 4.2). Since $|E_c|$ is no more than the number of VNR edges, the branching factor of the CT is finite. Moreover, a constraint recorded by any CT node either prohibits a VNR vertex from being mapped to an SN vertex or prohibits a VNR edge from utilizing an SN edge. Hence, there are only a finite number of such CT constraints. From Line 19 of Algorithm 1, we observe that a child CT node adds one such CT constraint to the set of its CT constraints inherited from its parent CT node. Also, Algorithm 1 obviates the need to create the child CT node if the added constraint is already part of the parent CT node. Hence, the set of CT constraints grows monotonically in size along any branch of the CT but can accumulate only a finite number of constraints. This means that the CT has a finite depth and—because of its finite branching factor—a finite size. □

Theorem 4.6. *For $w \geq 1$, VNE-CBS is a complete algorithm. That is, it returns a solution if one exists and correctly identifies an unsolvable VNE instance.*

Proof. Assume that a feasible VNE mapping p exists. There are four possible cases. First, VNE-CBS could expand a goal CT node that yields p, in which case it indeed returns a solution, namely p. Second, VNE-CBS could expand a goal CT node that yields a different solution, in which case it still returns a solution albeit different from p. Third, VNE-CBS could return "No Solution" on Line 31 of Algorithm 1, a case that we show by contradiction is impossible. This can happen only if *FOCAL* is empty on Line 9. Since *FOCAL* contains at least the minimum cost CT node in *OPEN* after every CT node expansion, this means that *OPEN* is also empty. However, this is impossible since, from Lemma 4.2, we know that *OPEN* always contains at least one CT node that permits p. Fourth, VNE-CBS could indefinitely expand CT nodes without returning either a solution or "No Solution". However, this is impossible since, from Lemma 4.5, we know that VNE-CBS expands only a finite number of CT nodes. From the analyses of the above cases, it follows that VNE-CBS returns a solution if one exists. Similarly, if the VNE instance is unsolvable, VNE-CBS returns "No Solution" on Line 31, since, otherwise, it would have to expand an infinite number of CT nodes in the loop of Lines 9–30, contradicting Lemma 4.5. □

5 iVNE-CBS

In this section, we further improve VNE-CBS by incorporating enhancements in both its high-level and low-level searches. These enhancements are inspired by analogous enhancements of CBS in the MAPF domain. They include disjoint splitting and bypassing conflicts in the high-level search and conflict avoidance tables, true cost heuristics, and more efficient duplicate detection in the low-level search. The incorporation of these enhancements culminates in the development of the iVNE-CBS algorithm, presented in Algorithm 3.

5.1 Enhancements in the High-Level Search

In this subsection, we describe the two major enhancements of the high-level search of iVNE-CBS. The first one is referred to as disjoint splitting, and the second one is referred to as bypassing conflicts. Both of them have been successfully used in the MAPF domain (Boyarski et al. 2015; Li, Harabor, et al. 2019).

Algorithm 3 shows the high-level search of iVNE-CBS with the blue lines highlighting the enhancements. (The black lines provide the backdrop of VNE-CBS in Algorithm 1.)

5.1.1 Disjoint Splitting. The idea of disjoint splitting (Li, Harabor, et al. 2019) is based on the observation that, when a conflict is resolved in the high-level search, the child CT nodes may often have overlaps in their search spaces. Disjoint splitting avoids these overlaps by using a conflict-resolution strategy based on constraint propagation. In iVNE-CBS, we apply disjoint splitting to resolve vertex conflicts efficiently.

Figure 4 illustrates the application of disjoint splitting for resolving vertex conflicts in the CBS framework for the VNE problem. The first panel shows a VNR containing three vertices and two edges that need to be embedded onto the SN shown below it. The second panel shows the resolution of a vertex conflict in the root CT node N without disjoint splitting. N has the VNE mapping A–1–4–B and A–3–5–C for the VNR edges A–B and A–C, respectively. It contains a type-1 vertex conflict $(A, 1, A, 3)$, where the VNR vertex A is mapped to the two SN vertices 1 and 3. VNE-CBS uses a non-disjoint splitting for resolving this conflict. It splits the CT node N by creating two child CT nodes, N_0 with a negative constraint $\overline{(A, 1)}$ and N_1 with a negative constraint $\overline{(A, 3)}$. At N_0, VNE-CBS replans the path A–1–4–B to A–2–4–B since the original path uses A–1, which is prohibited by the newly added negative constraint. Similarly, at N_1, it replans the path A–3–5–C to A–2–5–C. A path marked in bold indicates a path updated according to the constraints imposed by the high-level search. VNE-CBS splits N_0 to resolve the vertex conflict $(A, 2, A, 3)$ and creates two child CT nodes N_{00}, where the first path is replanned, and N_{01}, where the second path is replanned. Both N_{00} and N_{01} contain a feasible VNE mapping. Similarly, VNE-CBS

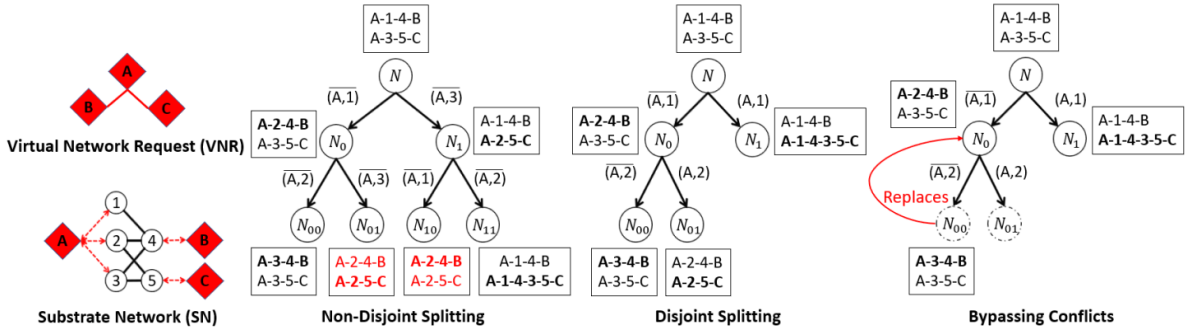

Fig. 4. An example of finding a VNE mapping in the CBS framework with non-disjoint splitting, disjoint splitting, and bypassing conflicts.

splits N_1 to resolve the vertex conflict $(A, 1, A, 2)$ and creates two child CT nodes N_{10} and N_{11}. Both N_{10} and N_{11} contain a feasible VNE mapping. However, the CT nodes N_{01} and N_{10} have the same set of constraints $\{\overline{(A, 1)}, \overline{(A, 3)}\}$. Thus, there is a duplication of search effort in their CT subtrees, shown by the two solutions marked in red.

Disjoint splitting can avoid such redundant search effort: For a type-1 vertex conflict (v^r, v_1^s, v^r, v_2^s), we choose an SN vertex $v_k^s \in \{v_1^s, v_2^s\}$ randomly and create two child CT nodes, one with a negative constraint $\overline{(v^r, v_k^s)}$, stopping v^r from being mapped to v_k^s and one with a positive constraint (v^r, v_k^s), forcing v^r to be mapped to v_k^s. The complementary negative and positive constraints do not allow for any overlaps in the search space of the two subtrees. Similarly, for a type-2 vertex conflict (v_1^r, v^s, v_2^r, v^s), we choose a VNR vertex $v_k^r \in \{v_1^r, v_2^r\}$ randomly and create two child CT nodes, one with a negative constraint $\overline{(v_k^r, v^s)}$, stopping v_k^r from being mapped to v^s and one with a positive constraint (v_k^r, v^s), forcing v_k^r to be mapped to v^s. In the subtree with the negative constraint, the low-level search maps the affected VNR vertex to a new SN vertex. In the subtree with the positive constraint, the low-level search maps the affected VNR vertex to the specified SN vertex and ignores all other possibilities.

The third panel shows the resolution of the same type-1 vertex conflict $(A, 1, A, 3)$ with disjoint splitting. It starts with the same root CT node N and splits on the vertex conflict $(A, 1, A, 3)$ by creating two child CT nodes N_0 and N_1 corresponding to VNR vertex A and SN vertex 1. At N_0, it replans A–1–4–B to A–2–4–B to satisfy the newly added negative constraint $\overline{(A, 1)}$. At N_1, it replans A–3–5–C to A–1–4–3–5–C to satisfy the newly added positive constraint $(A, 1)$. Next, it splits N_0 to resolve the vertex conflict $(A, 2, A, 3)$.

5.1.2 Bypassing Conflicts. Bypassing conflicts refers to another kind of conflict-resolution strategy. In the MAPF domain, it modifies the paths of the agents that are involved in a chosen conflict of a CT node instead of splitting that CT node (Boyarski et al. 2015). We use the same strategy in the VNE domain. When we expand a CT node N and generate its child CT nodes, if there exists a child CT node N_C such that $N_C.cost = N.cost$ and its number of conflicts is smaller than that of N, we replace the paths in N with the paths in N_C and do not generate any of its child CT nodes. If there is no such child CT node N_C, we split the CT node N as usual. Bypassing conflicts often reduces the number of CT node expansions necessary to find a solution. It can also be applied to focal search at the high level by modifying the criterion $N_C.cost = N.cost$ to $N_C.cost \leq w \cdot N.cost$ when choosing the child CT node.

In Figure 4, the last panel shows an example of bypassing conflicts, in which a CT node replaces its parent CT node when it has fewer conflicts and qualifies for expansion. iVNE-CBS with disjoint splitting finds a conflict $(A, 2, A, 3)$ at N_0 and starts to split on it. When it is about to create the child CT node N_{00}, it observes that $N_{00}.cost = N_0.cost$ since the path lengths do not change. Therefore, the condition $N_{00}.cost \leq w \cdot N_0.cost$ is satisfied. Moreover, the number of conflicts in $N_{00}.mapping$ is smaller than that in $N_0.mapping$. The procedure therefore replaces $N_0.mapping$ with $N_{00}.mapping$ and discards N_{00} and N_{01}. The new $N_0.mapping$ encodes a feasible solution.

5.1.3 Pseudocode. Algorithm 3 shows the high-level search of iVNE-CBS, that combines the described enhancements. The high-level enhancements are shown in blue. Here, we only explain these enhancements since the rest of the pseudocode is the same as that of VNE-CBS in Algorithm 1.

On Line 3, iVNE-CBS precomputes a table of true cost heuristics for the low-level search. This is an enhancement in the low-level search, which is explained in the next subsection. On Lines 16–17, iVNE-CBS generates a set of constraints *Cons* to resolve a conflict in $N_T.mapping$. When resolving conflicts, iVNE-CBS prefers to resolve vertex conflicts before bandwidth capacity conflicts, as described in Section 4.2. If disjoint splitting is enabled, the resolution of a vertex conflict is done by generating a negative and a positive constraint, as described in Figure 4. In a subtree with positive constraints, the low-level search is adapted to satisfy them, as explained in the next subsection.

On Lines 25–30, iVNE-CBS uses bypassing conflicts if it is enabled. If $N_C.cost \leq w \cdot N_T.cost$ and $N_C.num\_conf < N_T.num\_conf$, it replaces the mapping, cost, and the number of conflicts of N_T with those of N_C and discards all generated child CT nodes of N_T. If bypassing conflicts is not enabled or if the condition on Line 25 is not satisfied, iVNE-CBS splits N_T as in VNE-CBS.

5.2 Enhancements in the Low-Level Search

In this subsection, we describe the improved low-level search of iVNE-CBS. As described previously, it finds the shortest path from a fictitious start vertex v_1^f to a fictitious goal vertex v_2^f in G^m respecting $N.constraints$, where N is the corresponding high-level CT node. The fictitious vertices v_1^f and v_2^f represent the VNR vertices v_1^r and v_2^r, respectively.

The first enhancement is referred to as true cost heuristics. It is a precomputed table of shortest hop distances from all vertices to the goal vertices under the assumption that $N.constraints = \emptyset$. When, in fact, $N.constraints \neq \emptyset$, these precomputed distances serve as admissible heuristic estimates in the low-level search. The precomputation is done by running breadth-first searches from each goal vertex (such as v_2^f) to every vertex in G^m and storing the distances in a lookup table h_table (Line 3 of Algorithm 3). True cost heuristics have been successfully used in the MAPF domain (Sharon et al. 2015; Standley 2010). They are particularly effective in the VNE domain since G^m is normally an abstract graph on which other commonly used heuristic functions, such as the Manhattan distance between two vertices, are undefined. Therefore, while VNE-CBS does not use any heuristic function in the low-level search, iVNE-CBS has a good heuristic guidance from the precomputed distances. *OPEN* in the low-level search of iVNE-CBS is now sorted by the f-value, that is, the sum of the g-value and the true cost heuristic. This often focuses the search and yields smaller runtimes.

The second enhancement is the use of a Conflict Avoidance Table (CAT). The low-level search typically has to choose between several low-level nodes with the same minimum f-value for expansion. This is true in both the MAPF and the VNE domains. In such cases, a CAT is used to break ties in favor of nodes with fewer conflicts. Thus, the path returned by the low-level search is less likely to cause conflicts with other paths (Standley 2010). In the VNE domain, iVNE-CBS uses a single CAT for both vertex conflicts and bandwidth capacity conflicts. This leads to a tie-breaking rule that is more effective than the one used in VNE-CBS.

Algorithm 3 High-Level Search of iVNE-CBS.

1: Input: G^s, G^r, w
2: $G^m \leftarrow$ augmented graph for G^s and G^r
3: Precompute the true cost heuristic table *h_table*
4: $N_R \leftarrow$ *empty CT node*
5: $N_R.constraints \leftarrow \emptyset$
6: $N_R.mapping \leftarrow$ low-level paths found in G^m for all VNR edges using the low-level search
7: $N_R.cost \leftarrow \texttt{cost}(N_R.mapping)$
8: $N_R.num\_conf \leftarrow$ number of conflicts in $N_R.mapping$
9: $OPEN = FOCAL = \{N_R\}$
10: **while** $FOCAL \neq \emptyset$ **do**
11: $cost_{old} \leftarrow OPEN.top().cost$
12: $N_T \leftarrow FOCAL.top()$
13: Remove N_T from *OPEN* and *FOCAL*
14: **if** $N_T.num\_conf = 0$ **then**
15: **return** $N_T.mapping$ as the solution
16: $Conf \leftarrow$ first conflict found in $N_T.mapping$
17: $Cons \leftarrow$ (disjoint splitting) constraints for resolving $Conf$
18: **for** $C \in Cons$ **do**
19: $N_C \leftarrow$ copy of N_T
20: Add C to $N_C.constraints$
21: Update the low-level paths in $N_C.mapping$ to accommodate the constraint C
22: **if** update is successful **then**
23: $N_C.cost \leftarrow \texttt{cost}(N_C.mapping)$
24: $N_C.num\_conf \leftarrow$ number of conflicts in $N_C.mapping$
25: **if** $N_C.cost \leq w \cdot N_T.cost$ and $N_C.num\_conf < N_T.num\_conf$ **then**
26: $N_T.mapping \leftarrow N_C.mapping$
27: $N_T.cost \leftarrow N_C.cost$
28: $N_T.num\_conf \leftarrow N_C.num\_conf$
29: Discard all generated child CT nodes of N_T
30: Go to Line 14
31: $OPEN \leftarrow OPEN \cup \{N_C\}$
32: **if** $N_C.cost \leq w \cdot cost_{old}$ **then**
33: $FOCAL \leftarrow FOCAL \cup \{N_C\}$
34: $cost_{new} \leftarrow OPEN.top().cost$
35: **for** $N \in OPEN$ **do**
36: **if** $w \cdot cost_{old} < N.cost \leq w \cdot cost_{new}$ **then**
37: $FOCAL \leftarrow FOCAL \cup \{N\}$
38: **return** "No Solution"

A third enhancement is a more efficient implementation of duplicate detection. VNE-CBS only checks if a node has been previously expanded. However, a duplicate child node with the same f-value can still be generated and added to *OPEN*, resulting in a duplication of search effort downstream. In iVNE-CBS, we avoid this issue by checking for child nodes that duplicate previously generated nodes. We record the generated child nodes with their

f-values and do the following: If a child node has been previously generated, we retrieve the recorded f-value for comparison. If the f-value of the child node is smaller than the recorded f-value, we allow the generation of the child node and update the recorded f-value. Otherwise, we prune the child node. This implementation reduces the size of *OPEN*, which makes operations on it faster.

Another new feature of the low-level search of iVNE-CBS is its ability to support positive constraints in $N.constraints$. This is done when it generates the neighbors of a node while searching for a path from v_1^f to v_2^f for a VNR edge (v_1^r, v_2^r), where v_1^f and v_2^f are the fictitious vertices that represent v_1^r and v_2^r, respectively. When generating the neighbors of v_1^f, it returns the SN vertices v^s that satisfy $\text{CPU}(v^s) \geq \text{CPU}(v_1^f)$. If there is a positive constraint on v_1^f, it returns only the SN vertex in that positive constraint as its neighbor. When generating the neighbors of an SN vertex $v_1^s \in V^s$, it includes the SN vertices $v_2^s \in V^s$ such that $\text{BW}(e^s) \geq \text{BW}(e^r)$, where $e^s = (v_1^s, v_2^s)$ and $e^r = (v_1^r, v_2^r)$. It includes v_2^f if $\text{CPU}(v_1^s) \geq \text{CPU}(v_2^f)$. If there is a positive constraint on v_2^f, it includes v_2^f in the neighbors only when v_1^s is the SN vertex specified in that positive constraint.

5.3 Theoretical Properties of iVNE-CBS

In this subsection, we prove some theoretical properties of iVNE-CBS, analogous to those of VNE-CBS. We assume that the precomputation of the true cost heuristic table on Line 3 and disjoint splitting on Line 17 of Algorithm 3 are always enabled.[5] However, we split our analysis of iVNE-CBS into the two cases in which bypassing conflicts on Lines 25–30 is either enabled or disabled. From Lines 14–15, it is easy to observe that iVNE-CBS is sound in both cases. Below, we prove optimality, suboptimality, and completeness results for iVNE-CBS. The proofs of these results utilize the fact that the enhancements in the low-level search do not change the theoretical properties in any way, since the path-coordination task is delegated to the high-level search.

THEOREM 5.1. *For $w = 1$, with or without bypassing conflicts, a solution returned by iVNE-CBS is optimal.*

PROOF. We first prove that, for a CT node N, both the high-level and low-level searches of iVNE-CBS maintain the optimality of $N.cost$ under $N.constraints$. Although the low-level search of iVNE-CBS uses A^* search and other enhancements, it is identical to the low-level search of VNE-CBS in that it finds a shortest SN path to implement each VNR edge under $N.constraints$ independently. Hence, the low-level search of iVNE-CBS ensures the optimality of $N.cost$ under $N.constraints$. In the high-level search of iVNE-CBS, $N.cost$ is generally subject to change under bypassing conflicts, where Line 27 of Algorithm 3 replaces $N_T.cost$ with $N_C.cost$. However, for $w = 1$, these costs are identical and the optimality of $N_T.cost$ under $N_T.constraints$ is maintained.

From the above arguments and the fact that the set of feasible VNE mappings that satisfy $N.constraints$ is a subset of the set of VNE mappings that satisfy $N.constraints$, it follows that Lemma 4.1 continues to hold for iVNE-CBS with or without bypassing conflicts.

Using the same context and notation as in the proof of Lemma 4.2, we now prove that it also continues to hold for iVNE-CBS with or without bypassing conflicts. To prove this, we show that both disjoint splitting and bypassing conflicts preserve Lemma 4.2. For a type-1 vertex conflict (v^r, v_1^s, v^r, v_2^s), disjoint splitting generates two child CT nodes N_1 and N_2 with N_1 recording the constraint $\overline{(v^r, v_k^s)}$ and N_2 recording the constraint (v^r, v_k^s), for $v_k^s \in \{v_1^s, v_2^s\}$. For proof by contradiction, assume that p is not permitted by both N_1 and N_2. Since p is permitted by N but not by N_1, it must violate the constraint $\overline{(v^r, v_k^s)}$, that is, p maps v^r to v_k^s. Similarly, p must also violate the constraint (v^r, v_k^s), that is, p does not map v^r to v_k^s, yielding a contradiction. A similar argument shows that disjoint splitting on a type-2 vertex conflict also preserves Lemma 4.2. Moreover, bypassing conflicts also preserves Lemma 4.2, since it only replaces the mapping and the associated metrics of a parent CT node N_T

[5]The analyses of iVNE-CBS without one or both of these enhancements are omitted since they are simpler than the analysis presented here.

with those of a child CT node N_C without changing $N_T.constraints$. Hence, if p is permitted by N_T before the replacement, it is also permitted by N_T after the replacement.

Now consider any feasible VNE mapping p. Let its cost be $p.cost$. Because Lemma 4.2 continues to hold with or without bypassing conflicts, we know that all feasible VNE mappings are permitted at all stages by at least one CT node in *OPEN*. Let $N(p)$ be a CT node that permits p at the time when a goal CT node N_G is chosen for expansion. When this happens, iVNE-CBS returns a solution of cost $N_G.cost$. Although bypassing conflicts replaces $N_T.cost$ with $N_C.cost$ on Line 27 of Algorithm 3, the two costs are identical for $w = 1$. Hence, iVNE-CBS, with or without bypassing conflicts, expands CT nodes in a best-first manner. This means that $N_G.cost \leq N(p).cost$. Also, from Lemma 4.1, $N(p).cost$ is no more than $minCost(CV(N(p)))$, which in turn is no more than $p.cost$, since $p \in CV(N(p))$. Together, we have $N_G.cost \leq N(p).cost \leq p.cost$. Therefore, the cost of the solution returned by iVNE-CBS, with or without bypassing conflicts, is no more than the cost of any other feasible VNE mapping. □

Theorem 5.2. *For $w > 1$, without bypassing conflicts, a solution returned by iVNE-CBS is w-suboptimal.*

Proof. In the high-level search of iVNE-CBS, $N.cost$ is generally subject to change under bypassing conflicts. For $w > 1$, Lines 25 and 27 of Algorithm 3 indicate that $N_T.cost$ can be replaced by a larger $N_C.cost$. Hence, Lemma 4.1 is not necessarily preserved under bypassing conflicts, since $N_T.cost$ could grow larger than $minCost(CV(N_T))$. However, without bypassing conflicts, Lemma 4.1 is still preserved, since disjoint splitting by itself does not replace $N.cost$ for any CT node N. Therefore, without bypassing conflicts, Lemma 4.1 can be invoked as in the proof of Theorem 5.1: Using the same context and notation, this yields the inequality $N(p).cost \leq p.cost$. Moreover, $N_G.cost \leq w \cdot \tilde{N}.cost$, where $\tilde{N}$ is the CT node of minimum cost in *OPEN* at the time that N_G is chosen for expansion by the focal search procedure of iVNE-CBS. Since $\tilde{N}$ has the minimum cost, $\tilde{N}.cost \leq N(p).cost$. Combining the above inequalities, we have $N_G.cost \leq w \cdot \tilde{N}.cost \leq w \cdot N(p).cost \leq w \cdot p.cost$. Therefore, the cost of the solution returned by iVNE-CBS without bypassing conflicts is no more than w times the cost of any other feasible VNE mapping. □

Theorem 5.3. *For $w \geq 1$, with or without bypassing conflicts, iVNE-CBS is a complete algorithm. That is, it returns a solution if one exists and correctly identifies an unsolvable VNE instance.*

Proof. First, we show that Lemma 4.5 continues to hold for iVNE-CBS with or without bypassing conflicts. The branching factor of the CT pertains to conflict resolution. Compared to VNE-CBS, disjoint splitting in iVNE-CBS introduces a new way of branching but still results in a branching factor of 2. Hence, the branching factor of the CT remains finite. The depth of the CT also remains finite: Compared to VNE-CBS, disjoint splitting in iVNE-CBS introduces positive constraints. However, the number of such constraints is also finite. Hence, as in the proof of Lemma 4.5, the set of CT constraints, which grows monotonically in size along any branch of the CT, can accumulate only a finite number of positive or negative constraints, retaining the finite depth of the CT. Moreover, bypassing conflicts in iVNE-CBS can only curtail the depth of the CT after a CT node expansion since it prevents an immediate increase in depth by replacing the paths in the parent CT node and discarding its child CT nodes. Algorithm 3 also obviates the need to create a child CT node for an added positive or negative constraint if the added constraint is already part of the parent CT node.

For $w = 1$, we already show in the third paragraph of the proof of Theorem 5.1 that Lemma 4.2 continues to hold for iVNE-CBS with or without bypassing conflicts. These arguments are also applicable for any $w \geq 1$. (The statement of Theorem 5.1 is conditioned on $w = 1$ to preserve Lemma 4.1.)

The proof of Theorem 4.6 uses only Lemmas 4.2 and 4.5, which are both preserved in our current context. Hence, the same arguments as in the proof of Theorem 4.6 suffice for our purpose, provided that we show that Algorithm 3 performs a finite amount of computation for each CT node expansion. With bypassing conflicts, an implicit loop is created between Lines 14 and 30 within each CT node expansion, the finiteness of which we will now prove.

In the loop between Lines 14 and 30 of Algorithm 3, a parent CT node N_T is replaced by a child CT node N_C. However, such replacements cannot continue indefinitely within the scope of a CT node expansion because, on Line 25, the number of conflicts in $N_C.mapping$ is required to be strictly less than the number of conflicts in $N_T.mapping$. Starting from a finite value of $N_T.num\_conf$ and decreasing its value by at least 1 with each replacement, we can have only a finite number of replacements. □

In this article, we implement bypassing conflicts as a simple greedy high-level search strategy. However, in the MAPF domain, it has been used with a much more precise relationship to lower bounds on the solution cost associated with CT nodes (Li, Ruml, et al. 2021). In future work, we will improve the implementation of bypassing conflicts in iVNE-CBS.

6 Experiments

In this section, we present our experimental results in two parts. In the first part, we evaluate VNE-CBS against five popular baseline algorithms: D-ViNE, R-ViNE (Chowdhury, Rahman, et al. 2009), G-SP (Zhu and H. Ammar 2006), G-MCF (M. Yu et al. 2008), and RW-MaxMatch-SP (Cheng et al. 2011), which have been described in Section 2.1. We do so since it is common practice in the VNE literature to compare new algorithms for the core VNE problem against these five baseline algorithms (Yan et al. 2020; H. Zheng et al. 2017). Furthermore, for this part of our experiments, we also employ the standard experimental setup used in previous works in the VNE literature (Chowdhury, Rahman, et al. 2009; M. Yu et al. 2008). In the second part, we evaluate iVNE-CBS against VNE-CBS and several baseline algorithms. For this part of our experiments, we increase the size of the SNs and the VNRs substantially to demonstrate the efficiency and scalability of iVNE-CBS.

We implement all algorithms in C++ and conduct our experiments in ViNEYard, a VNE simulator (Chowdhury, Rahman, et al. 2012). We experiment with many configurations of VNE-CBS and iVNE-CBS: We refer to them using various self-explanatory suffixes. We run our experiments on an AWS machine with 8 AMD EPYC 9R14 3.7 GHz CPUs and 16 GB RAM. For all algorithms, we set a runtime limit of 60 s on each VNE instance.

We use Waxman graphs (M. Waxman 1988) to generate VNE instances, following the commonly used methodology in the VNE literature (Fischer et al. 2013). Waxman graphs are characterized by two parameters, α and β, and are frequently used in VNE simulations since their topologies resemble communication networks.[6] A Waxman graph is created by placing a specified number of vertices uniformly at random in a 2-dimensional rectangular area with a specified size, after which each pair of vertices at distance d is joined by an edge with probability $\beta \cdot e^{-d/(\alpha L)}$, where L is the maximum distance between any two vertices. We generate our SNs and VNRs using Waxman graphs with different values of α and β.

6.1 Evaluation of VNE-CBS

In this subsection, we evaluate VNE-CBS in both the offline and the online settings. The optimal version of VNE-CBS uses $w = 1.0$. Depending on whether it implements the tie-breaking rule mentioned in Section 4.3 in the low-level search, it is referred to as VNE-CBS-w1.0-tie or VNE-CBS-w1.0. In our experiments, we also include w-suboptimal variants of VNE-CBS, with and without the tie-breaking rule, for $w \in \{1.01, 1.05, 1.1, 1.5, 2.0\}$.

We use 5 SNs in our experiments. Each SN is a randomly generated Waxman graph with 100 vertices in a 50×50 2-dimensional space and parameter values $\alpha = 0.2$ and $\beta = 0.5$. The CPU and bandwidth capacities of the SN vertices and edges are real numbers drawn uniformly at random ($\in_u$) from the interval $[50, 100]$. Each VNR is also a randomly generated Waxman graph with the same parameter values $\alpha = 0.2$ and $\beta = 0.5$. Following the experimental setup used in previous works (Chowdhury, Rahman, et al. 2009; M. Yu et al. 2008), we set the number of vertices in each VNR to an integer drawn uniformly at random from the interval $[2, 10]$. The VNR

[6] Waxman graphs also find applications in Internet topology modeling, performance analysis of network protocols, and simulation of router networks.

Table 1. An evaluation of VNE-CBS in the offline setting. The number of VNR vertices $\in_u [2, 10]$; the VNR vertex CPU requirements $\in_u [0, 20]$; and the VNR edge bandwidth requirements $\in_u [0, 50]$. ± values indicate the 95% confidence intervals.

Algorithm	# Solved	# Timeout	# No Sol	Runtime (s)	Cost	# CT Nodes
D-ViNE	10,000	0	-	1.89	223.94	-
				±0.02	±2.72	-
R-ViNE	10,000	0	-	1.90	224.77	-
				±0.02	±2.70	-
G-SP	10,000	0	-	0.02	273.09	-
				±0.00	±3.28	-
G-MCF	10,000	0	-	0.50	234.18	-
				±0.00	±2.70	-
RW-MaxMatch-SP	10,000	0	-	0.02	248.53	-
				±0.00	±2.89	-
VNE-CBS-w1.0	9,432	568	0	0.83	141.61	130.90
				±0.07	±1.50	±12.47
VNE-CBS-w1.0-tie	9,616	384	0	0.63	141.61	91.40
				±0.06	±1.50	±8.55

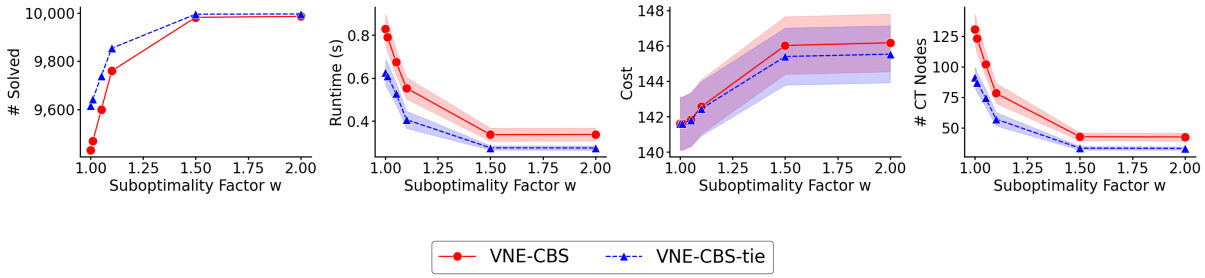

Fig. 5. The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 1. The 95% confidence intervals are shown as shaded areas.

vertices are located in the same 50×50 2-dimensional 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 $[0, 50]$. We generate 2,000 VNRs.

6.1.1 Offline Experiments. Tables 1, 2, 3, and 4 present our experimental results that test the various VNE algorithms in an offline setting. While Table 1 uses the 2,000 VNRs described above, Tables 2, 3, and 4 use different VNRs for stress testing: The higher demand levels in these VNRs provide insights into the efficiency and effectiveness of the VNE algorithms. 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 $[0, 50]$).

For each of the above settings, we map each of the 2,000 VNRs onto each of the 5 SNs, and therefore, there are 10,000 VNE instances provided to all VNE algorithms. Tables 1, 2, 3, and 4 show the number of solved VNE

Table 2. An evaluation of VNE-CBS in the offline setting. The number of VNR vertices $\in_u [2, 20]$; the VNR vertex CPU requirements $\in_u [0, 20]$; and the VNR edge bandwidth requirements $\in_u [0, 50]$. ± values indicate the 95% confidence intervals.

Algorithm	# Solved	# Timeout	# No Sol	Runtime (s)	Cost	# CT Nodes
D-ViNE	9,996	0	-	3.48	305.61	-
				±0.06	±5.62	-
R-ViNE	9,996	0	-	3.48	304.97	-
				±0.06	±5.57	-
G-SP	9,982	0	-	0.03	369.37	-
				±0.00	±6.74	-
G-MCF	9,996	0	-	0.82	314.57	-
				±0.01	±5.56	-
RW-MaxMatch-SP	10,000	0	-	0.03	593.61	-
				±0.00	±8.19	-
VNE-CBS-w1.0	5,334	4,666	0	2.04	188.66	192.31
				±0.16	±3.16	±15.77
VNE-CBS-w1.0-tie	5,813	4,187	0	1.72	188.66	166.60
				±0.14	±3.16	±14.87

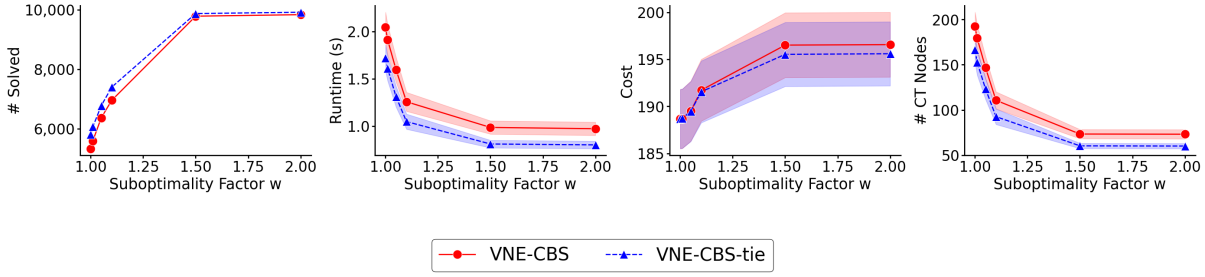

Fig. 6. The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 2. The 95% confidence intervals are shown as shaded areas.

instances (# Solved), the number of VNE instances for which the algorithm times out (# Timeout), and the number of VNE instances for which the algorithm returns the non-existence of solutions (# No Sol). We use "-" in the # No Sol columns if an algorithm cannot return the non-existence of solutions. All variants of VNE-CBS are complete, that is, # Solved + # Timeout + # No Sol is always equal to 10, 000. However, none of the other algorithms are complete, that is, as in Table 4, # Solved + # Timeout + # No Sol is not always equal to 10, 000.

Figures 5, 6, 7, and 8 present the results of the suboptimal variants of VNE-CBS with different values of w for the experimental settings in Tables 1, 2, 3, and 4, respectively. In the figures, the data points at $w = 1.0$ correspond to the performance of VNE-CBS-w1.0 and VNE-CBS-w1.0-tie. The tables and figures report results that are averaged over the VNE instances that are successfully solved by all algorithms. The reported metrics include the average runtime, the average cost, and the average number of CT nodes expanded during the high-level search (where relevant).

For the VNE instances solved successfully by all algorithms, all variants of VNE-CBS significantly outperform D-ViNE and R-ViNE in terms of the average runtime. They are also competitive with G-MCF on the average runtime, frequently outperforming it. In addition, all variants of VNE-CBS substantially outperform all other algorithms in terms of the average cost. However, for some VNE instances, both VNE-CBS algorithms with

Table 3. An evaluation of VNE-CBS in the offline setting. The number of VNR vertices $\in_u [2, 10]$; the VNR vertex CPU requirements $\in_u [0, 50]$; and the VNR edge bandwidth requirements $\in_u [0, 50]$. ± values indicate the 95% confidence intervals.

Algorithm	# Solved	# Timeout	# No Sol	Runtime (s)	Cost	# CT Nodes
D-ViNE	10,000	0	-	2.29	296.92	-
				±0.02	±3.25	-
R-ViNE	10,000	0	-	2.29	297.11	-
				±0.02	±3.23	-
G-SP	10,000	0	-	0.02	344.03	-
				±0.00	±3.75	-
G-MCF	10,000	0	-	0.72	306.87	-
				±0.01	±3.23	-
RW-MaxMatch-SP	9,995	0	-	0.01	326.65	-
				±0.00	±3.48	-
VNE-CBS-w1.0	9,378	622	0	0.87	217.63	109.34
				±0.08	±2.17	±10.01
VNE-CBS-w1.0-tie	9,553	447	0	0.66	217.63	78.58
				±0.06	±2.17	±7.55

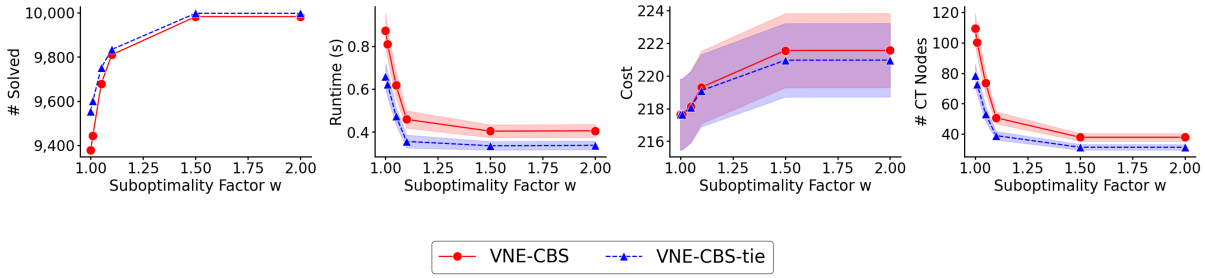

Fig. 7. The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 3. The 95% confidence intervals are shown as shaded areas.

$w = 1.0$ run more slowly than the five baseline algorithms and time out since they spend more time on finding a VNE mapping of minimum cost. As shown in the figures, compared to VNE-CBS-w1.0, the suboptimal variants with larger values of w solve substantially more instances, have substantially smaller average runtimes, do not substantially increase the average cost, and expand substantially fewer CT nodes on average. The same trends also hold for VNE-CBS-w1.0-tie and its suboptimal variants with larger values of w. Moreover, the VNE-CBS algorithms with the tie-breaking rule are better than the corresponding VNE-CBS algorithms without the tie-breaking rule with respect to all performance metrics.

6.1.2 Online Experiments. Table 5 presents our experimental results that test the VNE algorithms in an online setting, and Figure 9 presents the results of the suboptimal variants of VNE-CBS with different values of w for the same experimental setting. Here, VNRs arrive dynamically, and each successfully embedded VNR holds the CPU and bandwidth resources allocated to it on the SN until it departs at the end of its lifetime. The lifetime of a VNR is the period of time during which it stays in the network. (In this article, we do not consider the possibility of embedding a new VNR by reassigning the resources allocated to the previous VNRs. We leave this reconfiguration task for future work.) If an algorithm fails to embed a VNR within the time limit, it rejects the VNR and tries to embed the next one.

Table 4. An evaluation of VNE-CBS in the offline setting. The number of VNR vertices $\in_u [2, 10]$; the VNR vertex CPU requirements $\in_u [0, 20]$; and the VNR edge bandwidth requirements $\in_u [0, 80]$. ± values indicate the 95% confidence intervals.

Algorithm	# Solved	# Timeout	# No Sol	Runtime (s)	Cost	# CT Nodes
D-ViNE	9,997	0	-	1.91	319.10	-
				±0.02	±4.11	-
R-ViNE	9,996	0	-	1.90	320.03	-
				±0.02	±4.10	-
G-SP	9,966	0	-	0.03	428.91	-
				±0.00	±5.78	-
G-MCF	9,997	0	-	0.65	333.85	-
				±0.01	±4.08	-
RW-MaxMatch-SP	9,999	0	-	0.01	393.94	-
				±0.00	±5.15	-
VNE-CBS-w1.0	9,213	786	1	0.78	188.02	147.29
				±0.07	±2.12	±17.99
VNE-CBS-w1.0-tie	9,423	576	1	0.62	188.02	107.69
				±0.07	±2.12	±12.68

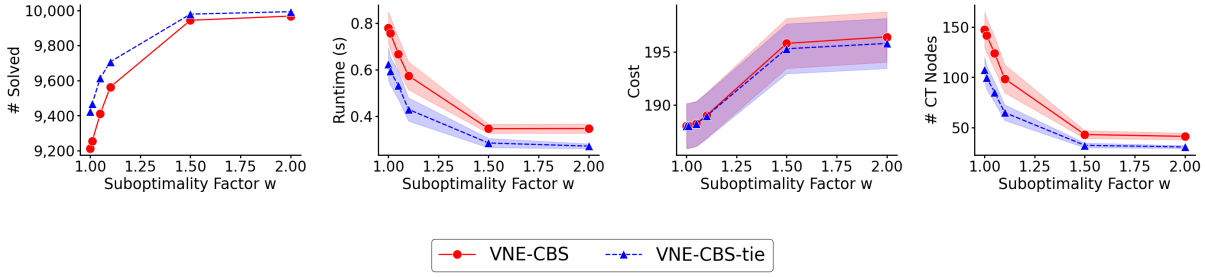

Fig. 8. The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 4. The 95% confidence intervals are shown as shaded areas.

Table 5 and Figure 9 use the same 2,000 VNRs as Table 1 and Figure 5. However, because of the online setting used in Table 5 and Figure 9, the VNRs arrive sequentially according to a Poisson process at an average rate of 4 VNRs per 100 timesteps. The lifetime of each VNR is generated from an exponential distribution with an average of 1,000 timesteps. Experimental results are gathered over 5 runs, each corresponding to one of the 5 SNs. In each run, the 2,000 VNRs arrive and depart in sequence and are embedded onto the SN chosen for that run. Each run ends when all VNRs have been processed and all successfully embedded VNRs have departed. Within each run, the acceptance ratio is the fraction of successfully embedded VNRs; the revenue and the cost are the sums of the revenues and the costs of all successfully embedded VNRs, respectively; the cost/revenue ratio is the cost divided by the revenue; and the runtime is the average runtime computed over the successfully embedded VNRs. Table 5 and Figure 9 report the averages of these metrics over the 5 runs.

All VNE algorithms have high average acceptance ratios. Consequently, they have very similar average revenues. But they differ remarkably in their average costs, and, therefore, also in their average cost/revenue ratios. The various VNE algorithms also differ in their average runtimes. All variants of VNE-CBS significantly outperform the five baseline algorithms in terms of the average cost and the average cost/revenue ratio. VNE-CBS-w1.0 and VNE-CBS-w1.0-tie have the best performance in terms of the average cost/revenue ratio since they are optimal.

Table 5. An evaluation of VNE-CBS in the online setting. The number of VNR vertices $\in_u [2, 10]$; the VNR vertex CPU requirements $\in_u [0, 20]$; and the VNR edge bandwidth requirements $\in_u [0, 50]$. Here, "Acceptance Ratio" is the average fraction of successfully embedded VNRs; "Rev" is the average revenue; "Cost" is the average cost; "Cost/Rev" is the average cost divided by revenue; and "Runtime" is the average runtime. All averages are computed over the 5 SNs described in the experimental setting. ± values indicate the 95% confidence intervals.

Algorithm	Acceptance Ratio	Rev	Cost	Cost/Rev	Runtime (s)
D-ViNE	0.987	293,680.8	491,697.3	1.68	1.95
	±0.00	±1,802.99	±4,200.39	±0.02	±0.13
R-ViNE	0.993	295,975.0	490,550.8	1.66	1.94
	±0.00	±1,512.50	±3,166.21	±0.01	±0.12
G-SP	0.967	285,677.1	604,714.6	2.12	0.02
	±0.02	±6,331.97	±14,615.87	±0.04	±0.00
G-MCF	0.969	286,858.4	494,049.2	1.72	0.54
	±0.02	±8,017.11	±14,402.76	±0.02	±0.01
RW-MaxMatch-SP	1.000	298,972.9	575,135.8	1.92	0.02
	±0.00	±0.00	±11,516.78	±0.04	±0.00
VNE-CBS-w1.0	0.953	243,231.1	243,322.1	1.00	4.00
	±0.01	±3,472.54	±3,454.66	±0.00	±0.90
VNE-CBS-w1.0-tie	0.959	276,522.1	276,684.1	1.00	3.52
	±0.01	±2,830.17	±2,808.75	±0.00	±0.39

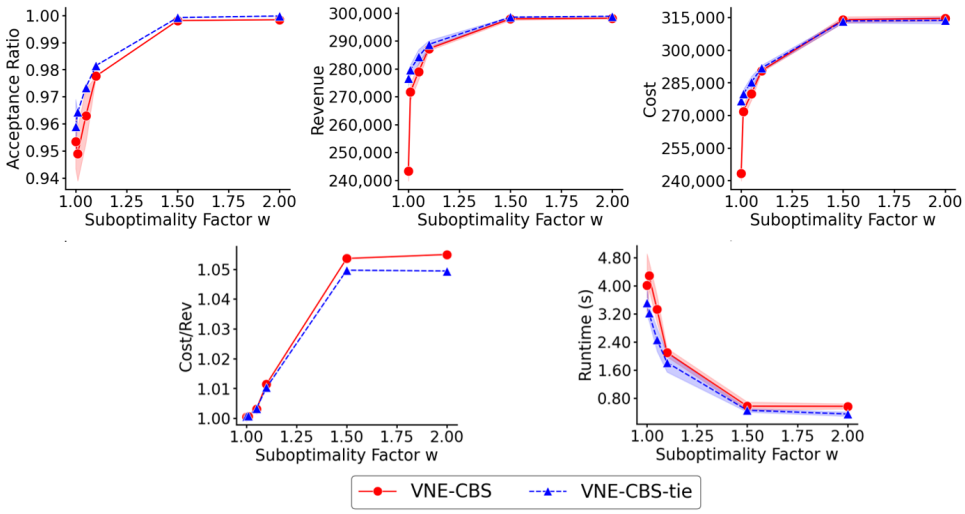

Fig. 9. The results of the suboptimal variants of VNE-CBS for the experimental setting in Table 5. The 95% confidence intervals are shown as shaded areas.

However, their average runtimes and average acceptance ratios are worse than those of the other VNE-CBS variants with larger w and the baseline algorithms, since they spend more time on finding a VNE mapping of minimum cost and are consequently more likely to time out. VNE-CBS-w2.0-tie, that is, VNE-CBS-tie with $w = 2.0$ in Figure 9, has a very high acceptance ratio of 0.999, excels on all other performance metrics, and has an average runtime of 0.35 s, making it significantly faster than most other VNE algorithms. Moreover, the

VNE-CBS algorithms with the tie-breaking rule are better than the corresponding VNE-CBS algorithms without the tie-breaking rule with respect to all performance metrics.

6.2 Evaluation of iVNE-CBS

In this subsection, we present our experimental results that compare iVNE-CBS against VNE-CBS, D-ViNE, R-ViNE, G-SP, G-MCF, and RW-MaxMatch-SP in both the offline setting and the online setting. In this set of experiments, we increase the scale of the SNs and the VNRs significantly beyond the scale that is traditionally used in the VNE literature: We do this to demonstrate the efficiency and scalability of iVNE-CBS.

The optimal version of iVNE-CBS uses $w = 1.0$. However, in our experiments, we also include suboptimal variants of both VNE-CBS and iVNE-CBS that use two other values of w, namely 1.5 and 2.0. Moreover, we invoke iVNE-CBS with and without disjoint splitting (indicated by the presence and absence, respectively, of "DS" in the label of the algorithm) and with and without bypassing conflicts (indicated by the presence and absence, respectively, of "BC" in the label of the algorithm). Compared to VNE-CBS, iVNE-CBS without disjoint splitting and bypassing conflicts incorporates enhancements only in the low-level search (denoted as "iVNE-CBS" without "DS" and "BC"). By comparing all these variants of VNE-CBS and iVNE-CBS, we conduct an ablation study to demonstrate the impact of the various enhancements that iVNE-CBS incorporates.

As in the previous section, we use Waxman graphs to generate the VNE instances. The SN topologies are randomly generated Waxman graphs with parameter values $\alpha = 0.1$ and $\beta = 0.3$. We generate 3 SNs, each with 500 vertices in a 100×100 grid space and with 3,694, 3,650, and 3,482 edges. The CPU and bandwidth capacities of the SN vertices and edges, respectively, are real numbers drawn uniformly at random from the interval $[50, 100]$. The VNR topologies are also randomly generated Waxman graphs with parameter values $\alpha = 0.2$ and $\beta = 0.3$. We set the number of VNR vertices to 10, 20, 30, 40, 50, 60, and 70, generating 1,000 VNRs for each setting of the number of VNR vertices, with an average of 7.05, 18.84, 35.55, 57.39, 86.16, 122.31, and 164.94 edges, respectively. The VNR vertices are located in the same 100×100 grid space as the SN vertices. The maximum allowed Euclidean distance for the geographical constraints on the VNR vertices is set to 15. The CPU and bandwidth requirements of the VNR vertices and edges are real numbers drawn uniformly at random from the intervals $[0, 20]$ and $[0, 50]$, respectively.

The SNs and VNRs used in our experiments are significantly larger than the SNs and VNRs used in previous works. Previous works use SNs with only around 100 vertices and 500 edges and VNRs with only around 10 vertices. Here, we use SNs with 500 vertices and around 3,500 edges and VNRs with up to 70 vertices and around 164 edges.

6.2.1 Offline Experiments. In the offline experiments, we map each of the 1,000 VNRs to each of the 3 SNs, for each setting of the number of VNR vertices. That is, we generate 3,000 VNE instances for each setting of the number of VNR vertices. Figures 10, 11, and 12 present the experimental results. Figures 10 and 11 demonstrate the impact of the high-level and low-level enhancements of iVNE-CBS, respectively. Based on these results, Figure 12 compares the best-performing variants of iVNE-CBS against the most scalable baseline algorithms. The figures do not show D-ViNE, R-ViNE, and G-MCF because of their poor performance. In particular, on VNE instances with 10 VNR vertices, D-ViNE and R-ViNE solve only 54 and 49 out of the 3,000 VNE instances, respectively, with average runtimes of 58.28 s and 58.44 s, respectively. They do not solve any of the larger VNE instances. On VNE instances with 10 VNR vertices, G-MCF solves 2,997 out of the 3,000 VNE instances with an average runtime of 22.12 s. However, on VNE instances with 20 VNR vertices, it solves only 875 out of the 3,000 VNE instances with an average runtime of 49.85 s. It does not solve any of the VNE instances with more than 20 VNR vertices.

Figure 10 shows the performance of VNE-CBS, iVNE-CBS (with only the low-level enhancements and without "DS" and "BC"), iVNE-CBS-DS, iVNE-CBS-BC, and iVNE-CBS-DS-BC for different values of w. The performance metrics include the number of solved VNE instances, the average runtime, the average cost of the solution, and

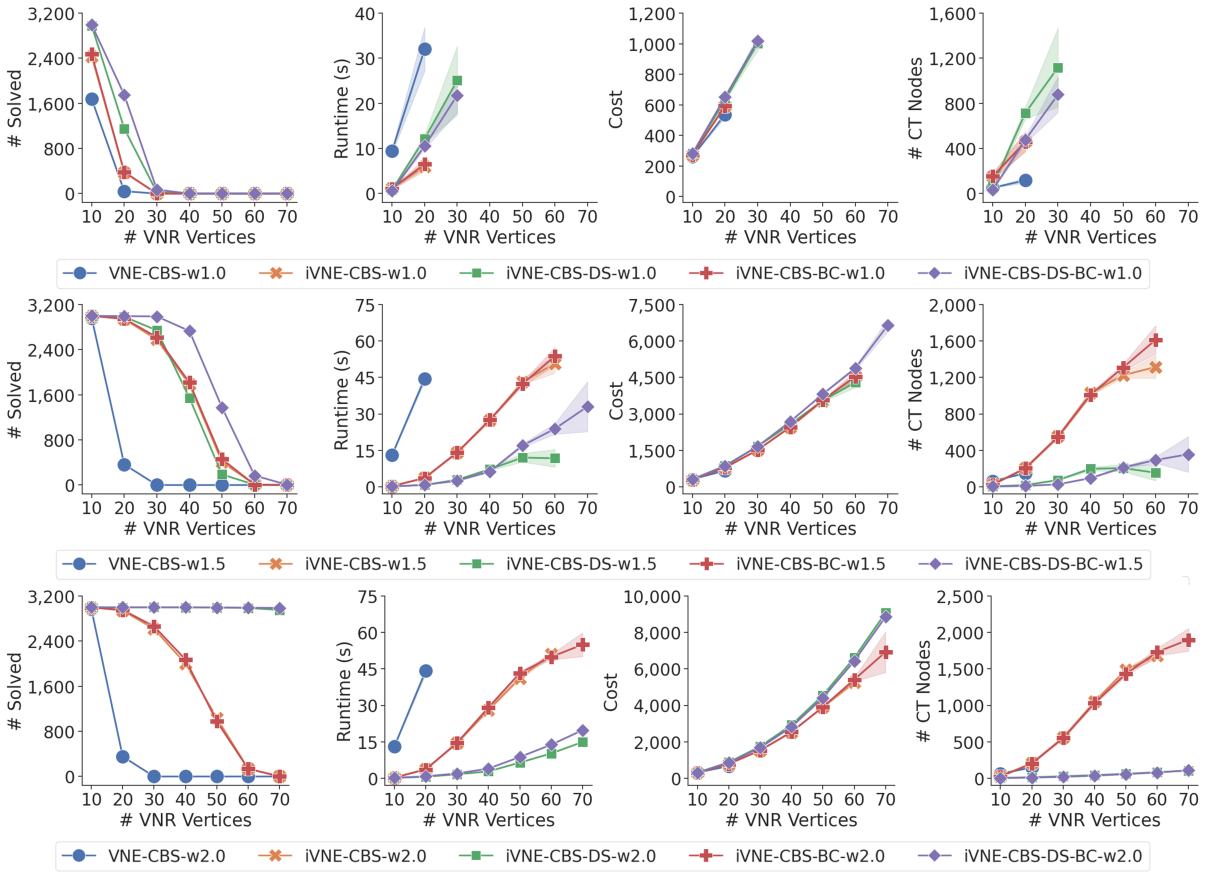

Fig. 10. The results of offline experiments that compare VNE-CBS and variants of iVNE-CBS with $w = 1.0$, 1.5, and 2.0. The 95% confidence intervals are shown as shaded areas.

the average number of CT node expansions required to find a solution. The number of solved VNE instances is the number of VNE instances that are successfully solved within the time limit. The other performance metrics are averaged over the VNE instances that are successfully solved (categorized by each setting of the number of VNR vertices). No data point is reported for an algorithm if it failed to solve any VNE instance.

First, we examine the impact of the low-level enhancements of iVNE-CBS on the number of solved VNE instances. Figure 10 shows that iVNE-CBS with only the low-level enhancements is able to outperform VNE-CBS for different values of w. Figure 11 demonstrates the reason for this performance difference. It shows that the low-level enhancements of iVNE-CBS substantially reduce the average number of low-level nodes expanded per CT node. This improves the runtime of the low-level search and, consequently, increases the number of solved VNE instances within the time limit.

Second, we examine the impact of the high-level enhancements of iVNE-CBS on the number of solved VNE instances. In general, iVNE-CBS algorithms that include one or more of the high-level enhancements significantly outperform VNE-CBS. They also generally solve more VNE instances with increasing values of w. Examined individually, disjoint splitting is an effective high-level enhancement since it reduces the average number of

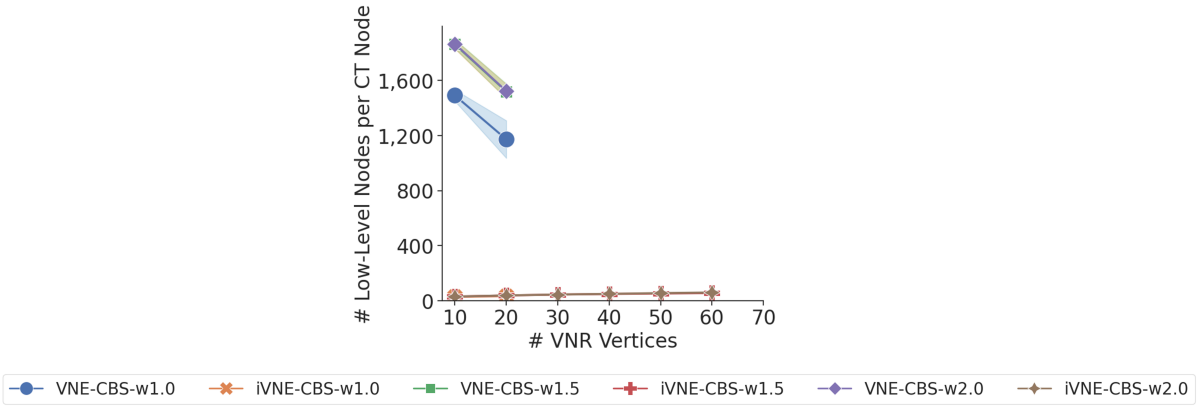

Fig. 11. The results of offline experiments that compare the average numbers of low-level nodes expanded per CT node for VNE-CBS and iVNE-CBS. The plot for VNE-CBS-w1.5 is hidden behind the plot for VNE-CBS-w2.0. The plots for iVNE-CBS-w1.0, iVNE-CBS-w1.5, and iVNE-CBS-w2.0 overlap at the bottom. The 95% confidence intervals are shown as shaded areas.

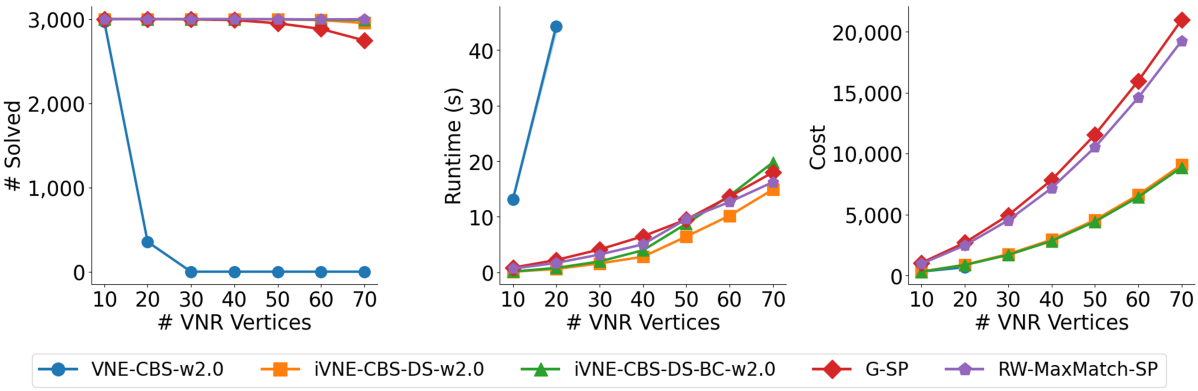

Fig. 12. The results of offline experiments that compare iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 against other baseline algorithms. In the first panel, the plots for iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 are hidden behind the plot for RW-MaxMatch-SP. For 70 VNR vertices, RW-MaxMatch-SP solves all 3,000 VNE instances, while iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 solve 2,954 and 2,990 VNE instances, respectively. In the second and third panels, the 95% confidence intervals are too small to be seen.

expanded CT nodes: iVNE-CBS-DS-w2.0 solves 2,954 VNE instances with 70 VNR vertices and significantly outperforms iVNE-CBS-w2.0. However, examined individually, bypassing conflicts is only a marginally effective high-level enhancement since it is a greedy strategy: iVNE-CBS-BC-w2.0 barely outperforms iVNE-CBS-w2.0. Nonetheless, iVNE-CBS-DS is slightly more effective with bypassing conflicts than without it: iVNE-CBS-DS-BC-w2.0 solves 46 VNE instances with 70 VNR vertices more than iVNE-CBS-DS-w2.0, increasing the number of solved VNE instances to 2,990 in this category.

Third, we examine the impact of the iVNE-CBS enhancements on the runtime and the cost of the solution produced. The average runtimes of all iVNE-CBS algorithms are smaller than those of VNE-CBS and decrease as the value of w increases. For the same value of w, the average costs of the solutions produced by all algorithms

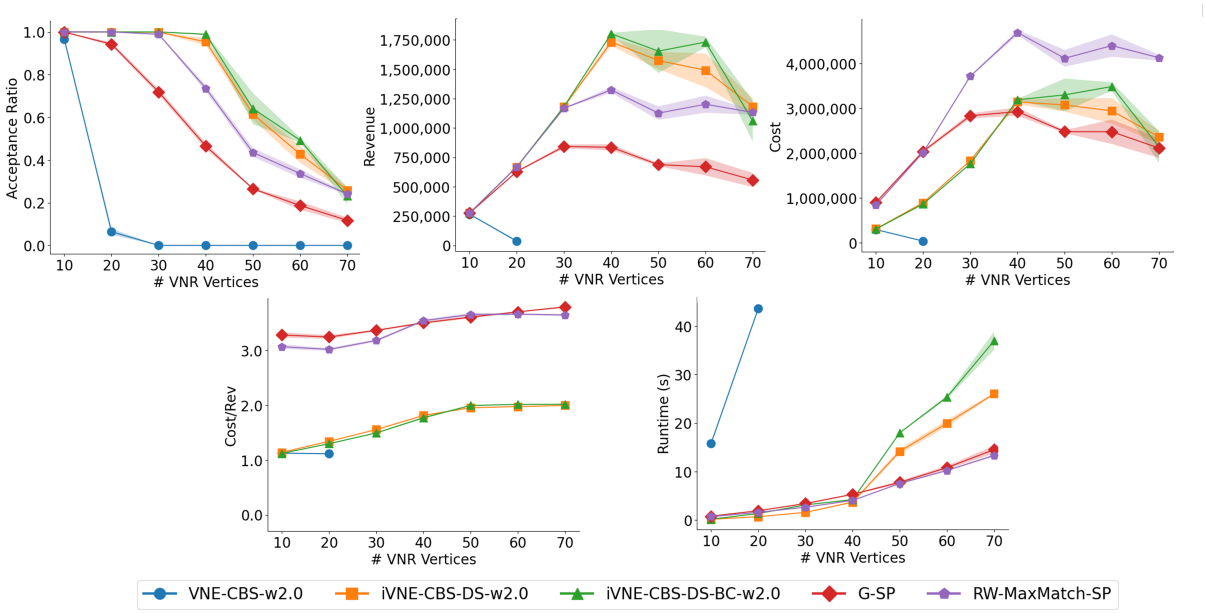

Fig. 13. The results of online experiments that compare iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 against the baseline algorithms. Here, "Acceptance Ratio" is the average fraction of successfully embedded VNRs; "Revenue" is the average revenue; "Cost" is the average cost; "Cost/Rev" is the average cost divided by revenue; and "Runtime" is the average runtime. All averages are computed over the 3 SNs described in the experimental setting. The 95% confidence intervals are shown as shaded areas.

are similar. In fact, for $w = 1.0$, VNE-CBS and all iVNE-CBS variants are theoretically guaranteed to produce optimal solutions. For larger values of w, VNE-CBS and all iVNE-CBS variants produce solutions of comparable costs in practice.

Overall, we can conclude that the low-level enhancements and disjoint splitting are the main contributors to the superior performance of iVNE-CBS compared to VNE-CBS. Bypassing conflicts can sometimes be beneficial when paired with disjoint splitting. In fact, iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 significantly outperform all other variants in terms of the number of solved VNE instances, the average runtime, and the average number of expanded CT nodes required to find a solution.

From the foregoing observations, we draw iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 as the two best CBS-based algorithms for comparison against G-SP and RW-MaxMatch-SP. Figure 12 shows this comparison on the performance metrics of the number of solved VNE instances, the average runtime, and the average cost. We also include VNE-CBS-w2.0 as a baseline algorithm in this comparison. We observe that the iVNE-CBS algorithms solve almost all of the VNE instances nearly as fast as G-SP and RW-MaxMatch-SP. However, the average costs of the solutions produced by the iVNE-CBS algorithms are significantly smaller than those produced by G-SP and RW-MaxMatch-SP. Moreover, unlike G-SP and RW-MaxMatch-SP, the iVNE-CBS algorithms have the theoretical properties discussed in Section 5.3.

6.2.2 Online Experiments. In the online setting, the experimental setup is the same as the one in Section 6.1.2, except that we use the 3 SNs and the VNRs generated for evaluating iVNE-CBS as described in Section 6.2. Hence, on the 1,000 VNRs generated for each setting of the number of VNR vertices, the experiments utilize 3 runs. Each

run embeds the 1,000 VNRs onto one of the 3 SNs. We choose $w = 2.0$ for all CBS-based algorithms since it yields the best performance in the offline experiments. For the iVNE-CBS variants, we choose iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 for comparison since they are the best CBS-based algorithms for solving VNE instances (as evidenced in Figure 10). They are evaluated against the same competing algorithms that appear in Figure 12.

Figure 13 presents the results on the performance metrics of the average acceptance ratio, the average revenue, the average cost, the average cost/revenue, and the average runtime. The acceptance ratio is averaged over the 3 runs. On this metric, the iVNE-CBS variants outperform VNE-CBS and the other baseline algorithms. Compared to the other algorithms, the iVNE-CBS variants generally allocate the SN resources more cost-effectively to previously accepted VNRs and are thus able to accommodate new VNRs more easily. The average revenue is the sum of the revenues of all successfully embedded VNRs averaged over the 3 runs. On this metric, the iVNE-CBS variants outperform VNE-CBS and the other baseline algorithms before doing slightly worse for 70 VNR vertices. The average cost is the sum of the costs of all successfully embedded VNRs averaged over the 3 runs. On this metric, the iVNE-CBS variants outperform RW-MaxMatch-SP despite accumulating more cost by solving more VNE instances. However, they are offset by G-SP for larger numbers of VNR vertices because G-SP does not solve many VNE instances in such settings. The average cost/revenue is the cost divided by the revenue averaged over the 3 runs. On this metric, the iVNE-CBS variants outperform G-SP and RW-MaxMatch-SP, indicating that they allocate the SN resources to accepted VNRs more cost-effectively. The average runtime is computed over the successfully embedded VNRs across the 3 runs. On this metric, G-SP and RW-MaxMatch-SP appear to do better than the iVNE-CBS variants. However, this is actually a result of the higher acceptance ratios of the iVNE-CBS variants. They have higher acceptance ratios because they solve hard VNE instances that are not solved by the other algorithms. For the same reason, they also have larger average runtimes.

7 Conclusions and Future Work

The VNE problem is an NP-hard problem in network resource management. While some network resources, such as the compute power, can be localized, other network resources, such as the communication bandwidth, can be distributed. The VNE problem models the proper orchestration of all kinds of network resources so as to support network virtualization as an emerging technology. With network virtualization, 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.

In this article, we presented a new suite of VNE algorithms based on the CBS framework. While we adopted the CBS framework because of its success in the MAPF domain, we also carefully adapted it to make it successful in the VNE domain. We presented two new VNE algorithms, namely VNE-CBS and iVNE-CBS, both of which can be invoked as either optimal or bounded-suboptimal algorithms.

We showed that, unlike many existing VNE algorithms, VNE-CBS is both complete and optimal. We also developed a bounded-suboptimal version of VNE-CBS that trades off optimality for increased efficiency. Through experiments, we showed that VNE-CBS substantially outperforms competing state-of-the-art VNE algorithms on various benchmark instances in both the offline and the online settings of the VNE problem.

iVNE-CBS improves on VNE-CBS with various enhancements in the high-level and low-level searches, such as disjoint splitting, bypassing conflicts, true cost heuristics, and CATs. These enhancements are inspired by similar enhancements in the MAPF domain. iVNE-CBS significantly outperforms VNE-CBS and other popular VNE algorithms in both the offline and the online settings and in terms of both runtime and solution quality. In fact, iVNE-CBS scales well to large VNE instances, much beyond the scale of the VNE instances previously studied in the literature. Its success supports the promise of cross-fertilizing ideas between the MAPF and the VNE domains.

Overall, our article serves as a case study in how successful algorithmic techniques developed in one research community can be applied to cornerstone problems in another research community. In particular, it shows how

MAPF technologies can be used in the VNE domain. In future work, we will continue to exploit this connection and generalize our algorithms to solve other variants of the VNE problem such as: (a) the VNE problem with path-splitting, where a VNR edge can be implemented as a collection of paths on the SN; and (b) the VNE problem with slice migration, where an embedding of a VNR can be reconfigured to accommodate new VNRs.

Acknowledgments

This research was supported by DARPA under grant number HR001120C0157 and by NSF under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, 2112533, and 2121028. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. government.

References

A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines. 2020. "5G Network Slicing Using SDN and NFV: A Survey of Taxonomy, Architectures and Future Challenges." *Computer Networks*.

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

E. Boyarski, A. Felner, G. Sharon, and R. Stern. 2015. "Don't Split, Try to Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding." In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*.

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

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

S. Choudhury, K. Solovey, M. J. Kochenderfer, and M. Pavone. 2020. "Efficient Large-Scale Multi-Drone Delivery Using Transit Networks." In: *Proceedings of the IEEE International Conference on Robotics and Automation*.

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

M. Chowdhury, M. R. Rahman, and R. Boutaba. 2012. "ViNEYard: Virtual Network Embedding Algorithms with Coordinated Node and Link Mapping." *IEEE/ACM Transactions on Networking*.

M. Chowdhury, M. R. Rahman, and R. Boutaba. 2009. "Virtual Network Embedding with Coordinated Node and Link Mapping." In: *Proceedings of the Twenty-Eighth Joint Conference of the IEEE Computer and Communications Societies*.

L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig. 2016. "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*.

N. Feamster, L. Gao, and J. Rexford. 2007. "How to Lease the Internet in Your Spare Time." *Computer Communication Review*.

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

O. Gordon, Y. Filmus, and O. Salzman. 2021. "Revisiting the Complexity Analysis of Conflict-Based Search: New Computational Techniques and Improved Bounds." In: *Proceedings of the Fourteenth International Symposium on Combinatorial Search*.

J. Li, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig. 2019. "Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search." In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*.

J. Li, W. Ruml, and S. Koenig. 2021. "EECBS: Bounded-Suboptimal Search for Multi-Agent Path Finding." In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*.

K. M. Dresner and P. Stone. 2008. "A Multiagent Approach to Autonomous Intersection Management." *Journal of Artificial Intelligence Research*.

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

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

P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi. 2018. "5G Wireless Network Slicing for eMBB, URLLC, and mMTC: A Communication-Theoretic View." *IEEE Access*.

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

M. Richart, J. Baliosian, J. Serrat, and J. Gorricho. 2016. "Resource Slicing in Virtual Wireless Networks: A Survey." *IEEE Transactions on Network and Service Management*.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. "Conflict-Based Search for Optimal Multi-Agent Pathfinding." *Artificial Intelligence*.

D. Silver. 2005. "Cooperative Pathfinding." In: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*.

T. S. Standley. 2010. "Finding Optimal Solutions to Cooperative Pathfinding Problems." In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence.*

R. Stern et al.. 2019. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks." In: *Proceedings of the Twelfth International Symposium on Combinatorial Search.*

The European Telecommunications Standards Institute. 2020. *5G; Management and Orchestration; Concepts, Use Cases and Requirements.* https://www.etsi.org/deliver/etsi_ts/128500_128599/128530/16.02.00_60/ts_128530v160200p.pdf.

Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li. 2020. "Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach with Graph Convolutional Networks." *IEEE Journal on Selected Areas in Communications.*

J. Yu and S. LaValle. 2013. "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs." In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence.*

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

H. Zheng, J. Li, Y. Gong, W. Chen, Z. Yu, Z. Zhan, and Y. Lin. 2017. "Link Mapping-Oriented Ant Colony System for Virtual Network Embedding." In: *Proceedings of the IEEE Congress on Evolutionary Computation.*

Y. Zheng, S. Ravi, E. Kline, S. Koenig, and T. K. S. Kumar. 2022. "Conflict-Based Search for the Virtual Network Embedding Problem." In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling.*

Y. Zheng, S. Ravi, E. Kline, L. Thurlow, S. Koenig, and T. K. S. Kumar. 2023. "Improved Conflict-Based Search for the Virtual Network Embedding Problem." In: *Proceedings of the Thirty-Second International Conference on Computer Communications and Networks.*

Y. Zhu and M. H. Ammar. 2006. "Algorithms for Assigning Substrate Network Resources to Virtual Network Components." In: *Proceedings of the Twenty-Fifth Joint Conference of the IEEE Computer and Communications Societies.*

A Table of Notation

(Shown on the next page.)

Table 6. Notation used in this article.

Notation	Definition
$G^s = (V^s, E^s, A^s_V, A^s_E)$	An SN
V^s	The set of SN vertices
E^s	The set of SN edges
A^s_V	A mapping from SN vertices to their attributes
A^s_E	A mapping from SN edges to their attributes
$\textsc{cpu}(v^s)$	The CPU capacity of an SN vertex v^s
$\textsc{loc}(v^s)$	The geographical location of an SN vertex v^s
$\textsc{bw}(e^s)$	The bandwidth capacity of an SN edge e^s
$G^r = (V^r, E^r, C^r_V, C^r_E)$	A VNR
V^r	The set of VNR vertices
E^r	The set of VNR edges
C^r_V	A mapping from VNR vertices to their demands
C^r_E	A mapping from VNR edges to their demands
$\textsc{cpu}(v^r)$	The CPU requirement of a VNR vertex v^r
$\textsc{loc}(v^r)$	The preferred geographical location of a VNR vertex v^r
$D(v^r)$	The maximum allowed distance from v^r's preferred geographical location to the location of the SN vertex it is mapped to
$\textsc{bw}(e^r)$	The bandwidth requirement of a VNR edge e^r
$\textsc{vne}(\cdot)$	The VNE mapping of a VNR component
$\textsc{vne}(v^r)$	The SN vertex that a VNR vertex v^r is mapped to
$\textsc{vne}(e^r)$	The SN path that a VNR edge e^r is mapped to
$\textsc{geodist}(\cdot,\cdot)$	The distance between two geographical locations
G^m	The augmented graph created from a VNR and an SN
$v^f \in V^f$	A fictitious vertex that represents a VNR vertex on G^m
(v^f_i, v^s_1)	A fictitious edge from a VNR vertex to an SN vertex on G^m
(v^r, v^s_1, v^r, v^s_2)	A type-1 vertex conflict where a VNR vertex v^r is mapped to two different SN vertices v^s_1 and v^s_2
$\overline{(v^r, v^s_1)}$	A negative constraint that stops v^r from being mapped to v^s_1
(v^r_1, v^s, v^r_2, v^s)	A type-2 vertex conflict where two VNR vertices v^r_1 and v^r_2 from the same VNR are mapped to the same SN vertex v^s
w	A user-specified suboptimality factor
N	A CT node
$N.mapping$	The set of paths (VNE mapping) associated with CT node N
$N.cost$	The cost of the VNE mapping associated with CT node N
$N.num\_conf$	The number of conflicts in the VNE mapping associated with CT node N
$N.constraints$	The set of accumulated constraints associated with CT node N
$Conf$	A conflict in the VNE mapping $N.mapping$
$Cons$	A set of constraints generated for resolving $Conf$
n_T	A top low-level search node retrieved from the low-level *OPEN* list
$CV(N)$	The set of all feasible VNE mappings that satisfy the constraints associated with CT node N
$minCost(CV(N))$	The minimum cost over all feasible VNE mappings in $CV(N)$
E_c	The set of VNR edges that utilize a common SN edge
p	A feasible VNE mapping
$N(p)$	A CT node that permits the feasible VNE mapping p
h_table	The precomputed table of heuristic values used by the low-level search
(v^r, v^s_1)	A positive constraint that forces v^r to be mapped to v^s_1
α, β	The parameters used to generate Waxman graphs
$\in_u$	Sampling uniformly at random

B Table of Acronyms

Table 7. Acronyms used in this article.

Acronym	Expansion
3GPP	3rd Generation Partnership Project
aaS	as a Service
BC	Bypassing Conflicts
BMaaS	Bare-Metal aaS
CAT	Conflict Avoidance Table
CBS	Conflict-Based Search
CT	Constraint Tree
DS	Disjoint Splitting
ECBS	Enhanced CBS
GT-ITM	Georgia Tech Internet Topology Model
IaaS	Infrastructure aaS
ILP	Integer Linear Programming
ISP	Internet Service Provider
iVNE-CBS	Improved VNE-CBS
LP	Linear Programming
MAPF	Multi-Agent Path Finding
MIP	Mixed Integer Programming
NSaaS	Network Slice aaS
PaaS	Platform aaS
PoP	Point of Presence
QoS	Quality of Service
RAN	Radio Access Network
SaaS	Service aaS
SN	Substrate Network
URLLC	Ultra-Reliable Low Latency Communications
VNE	Virtual Network Embedding
VNE-CBS	CBS for VNE
VNF	Virtual Network Function
VNR	Virtual Network Request

Received 20 January 2025; accepted 17 February 2026