Improved Conflict-Based Search for the Virtual Network Embedding Problem

Yi Zheng, Srivatsan Ravi, Erik Kline, Lincoln Thurlow, Sven Koenig and T. K. Satish Kumar
University of Southern California, Los Angeles, California, USA
Email: yzheng63@usc.edu, sravi@isi.edu, kline@isi.edu, lincoln@isi.edu, skoenig@usc.edu, tkskwork@gmail.com

***Abstract*—Virtualization is the mechanism of creating virtual representations of physical resources. It is now integrated into almost every facet of computing and is pervasive on the Internet: ranging from data center services and cloud computing services to services on our phones. The common goal for virtualization providers is to ensure that the physical resources are managed efficiently and effectively. This goal induces the Virtual Network Embedding (VNE) problem: the task of properly allocating the physical resources of a network to satisfy virtual requests for resources under various constraints while ensuring the quality of service and maximizing resource utilization. The VNE problem captures many resource allocation tasks arising in computer systems and computer networks. In this paper, we present Improved VNE-CBS (iVNE-CBS) as an efficient and effective algorithm for solving the VNE problem. iVNE-CBS builds on Conflict-Based Search (CBS), a heuristic search framework borrowed from the Multi-Agent Path Finding literature. We show that iVNE-CBS significantly outperforms popular baseline VNE algorithms: it scales to networks with several hundreds of vertices and thousands of edges, while also producing better-quality solutions.**

***Index Terms*—network virtualization, virtual network embedding, conflict-based heuristic search**

I. Introduction

Virtualization is a mainstay in the modern computer and networking ecosystem. It enables 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). 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. 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) requirements. This capability serves 5G Networks and Network Slice as a Service (NSaaS), as described in the 3GPP standard [1].

While virtualization provides a "logical" representation of physical resources leading to many benefits, it introduces an orchestration problem: the physical resources have to be managed efficiently and effectively in a shared environment that has many users and many kinds of constraints. This is often referred to as the Virtual Network Embedding (VNE) problem. It is essentially the task of properly allocating the physical resources of a network to satisfy virtual requests for resources under various constraints while ensuring the quality of service and maximizing resource utilization.

The VNE problem can also be understood as having to satisfy the requirements of a Virtual Network Request (VNR) by allocating the physical resources of a Substrate Network (SN) to it. A VNR is a graph in which vertices represent logical compute nodes and edges represent logical communication links between pairs of compute nodes. A proper VNE 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 various CPU and bandwidth capacity constraints.

The VNE problem captures many resource allocation tasks arising in computer systems and computer networks. In the 5G environment, a customer could 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 certain Point of Presence (PoP). 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 is NP-hard to solve optimally [2]. Previous works on the VNE problem have developed various kinds of algorithms. However, many of these algorithms either produce low-quality solutions, are unable to scale to large VNE instances, and/or do not have strong theoretical properties. VNE-CBS [3] is a recent solver that has been developed to address these drawbacks. It bears both the advantages of being a complete solver and producing optimal solutions. VNE-CBS also empirically outperforms other popular VNE algorithms.

VNE-CBS is inspired by the success of the Conflict-Based Search (CBS) framework in the Multi-Agent Path Finding (MAPF) domain. In the MAPF problem, multiple agents are required to plan collision-free paths in a shared environment with obstacles [4]. Combinatorially, both the MAPF and VNE problems 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 different in subtle ways. First, the MAPF problem has a temporal dimension that is absent from the VNE problem. Second, the MAPF problem

dominantly has mutual exclusion constraints between pairs of agents, whereas the VNE problem dominantly has capacity constraints on the CPU and bandwidth resources. Therefore, the crossover of heuristic search techniques between the two problem domains requires careful adaptation.

VNE-CBS is a CBS-based algorithm that is carefully adapted to the VNE problem. Similar to CBS for MAPF, it is a two-level search algorithm. The high-level search is a best-first search that resolves conflicts arising from resource contentions. The low-level search is a path-finding algorithm that allocates resources to each VNR vertex and edge under the constraints imposed by the high-level search node.

In this paper, we improve on VNE-CBS and present Improved VNE-CBS (iVNE-CBS). Compared to 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. This includes 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. With these enhancements, iVNE-CBS has the following properties: On the theoretical front, it retains the completeness of VNE-CBS and its guarantee to produce optimal solutions. On the experimental front, it efficiently solves VNE instances that are significantly larger than those presented in previous VNE literature. It easily outperforms VNE-CBS and other baseline VNE algorithms both in terms of efficiency and solution quality.

II. Background

In this section, we present some background literature on the VNE problem and the CBS framework in the MAPF domain.

A. 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. These mappings are required to satisfy various constraints, such as: (a) For each SN vertex, the sum of the CPU requirements of all VNR vertices mapped to it should be no greater than its available CPU capacity. (b) For each SN edge, the sum of the bandwidth requirements of all VNR edges that utilize it should be no greater than its available bandwidth capacity. Additional constraints may also be used on the VNE mapping. For example, each VNR vertex may be constrained to be mapped to an SN vertex within a certain geographical distance from a preferred geographical location, such as a data center. Another constraint that is popularly imposed on the VNE mapping is to assign different SN vertices to different VNR vertices from the same VNR [5].

The VNE problem and its more expressive variants are known to be NP-hard [6].

The VNE problem arises in both offline and online settings. In the offline version of the VNE problem, the set of VNRs to be independently embedded in the SN is known in advance. In the online version, the VNRs arrive in sequence at different times and stay in the network for an arbitrary duration of time. In this case, reconfiguration may be necessary, i.e., it may be necessary to backtrack on the embedding assignments made for past VNRs to be able to accommodate new VNRs when they arrive. Moreover, in the online version, multiple VNRs may hold network resources simultaneously and, therefore, VNR vertices from different VNRs are allowed to be mapped to the same SN vertex.

There are many quality metrics on a VNE mapping. These include the revenue, the cost, and the acceptance ratio. While the revenue quantifies the total resources demanded by a successfully embedded VNR, the cost quantifies the total resources meted out to it 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 online and offline settings, the acceptance ratio quantifies the fraction of successfully embedded VNRs among the total number of VNRs. A popular objective is to maximize the acceptance ratio while minimizing the cost.

Many algorithms have been proposed to solve the VNE problem and its variants. A compendium of existing algorithms can be found in several survey articles [7], [8]. 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 [9].

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. For example, G-SP and G-MCF are two such algorithms [2]. Another suite of algorithms are inspired by Google's Page Rank algorithm [10]. These algorithms use the Markov Random Walk model to rank VNR and SN vertices based on their resources and topological attributes. A subsequent matching process utilizes these ranks.

B. Multi-Agent Path Finding

The MAPF problem is specified by an undirected unweighted graph $G = (V, E)$ and a set of k agents $\{a_1, ..., a_k\}$, where a_i moves 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 a_i and a_j are at the same vertex v at timestep t. An edge conflict (a_i, a_j, u, v, t) occurs when a_i and a_j traverse the same edge (u, v) in opposite directions between timestep t and $t+1$. A solution to a MAPF problem is a set of paths without conflicts. One common

objective is to find a solution while minimizing the sum of costs of all agents [11]. Solving the MAPF problem optimally for this objective is NP-hard [12], [13]. Several variants of the MAPF problem have been proposed to model different real-world situations [4]. The MAPF problem and its variants have many real-world applications, including in video games [14], automated warehousing [15], and in multi-drone delivery [16].

C. Conflict-Based Search

CBS is a two-level heuristic search algorithm for solving the MAPF problem optimally [11]. On the high level, CBS performs a best-first search on a Constraint Tree (CT). Each CT node N contains a set of spatiotemporal constraints $N.constraints$ that are used to coordinate agents to avoid conflicts. Each CT node has a set of paths $N.paths$, one for each agent, that respects the constraints. The cost of a CT node N is the sum of the costs of paths in $N.paths$. The root CT node contains an empty constraint set and a set of shortest paths that may contain conflicts. When CBS expands a CT node, it first checks for conflicts in its paths. If there are none, the CT node is a goal CT node and CBS returns the paths as the solution. Otherwise, CBS chooses one of the conflicts and resolves it by splitting into two child CT nodes, each with an additional constraint prohibiting one agent from the conflict from using the conflicting vertex or edge at the conflicting timestep. CBS uses its low-level search (e.g., A^*) to replan the path of the agent to satisfy the new constraint. The fewer conflicts there are to resolve, the faster CBS terminates. A complexity analysis of CBS has been presented in [17]. CBS guarantees completeness by eventually exploring both ways of resolving each conflict and optimality of the generated solution by performing best-first search on both its high and low levels.

However, since solving a 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 the solution cost for runtime [18]. 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, like in A^*, whose nodes n are sorted by an admissible cost function $f(n) = g(n) + h(n)$ where $h(n)$ is an admissible heuristic function. Let $w > 1$ be a user-specified suboptimality factor and $f_{min} = \min_{n_i \in OPEN} f(n_i)$ be the minimum f-value 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 can be inadmissible) that estimates the number of hops from node n to a goal node. Unlike A^*, focal search always expands a node n with the minimum d-value in *FOCAL*. Let C^* be the optimal solution cost. Focal search guarantees that the returned solution cost 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 these 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 factor w to be used in its focal searches. ECBS(w) is w-suboptimal since it guarantees a solution with a cost that is no larger than $w \cdot C^*$ [18]. Thus, ECBS(w) with a reasonably small value of w has the flexibility of expanding CT nodes with fewer conflicts than the CT nodes chosen by CBS. This often makes ECBS(w) faster than CBS.

III. Problem Formulation

In this paper, we solve the VNE problem as defined in [9].

A. Substrate Network

We define an SN as an undirected graph $G^s = (V^s, E^s)$, where V^s is the set of SN vertices and E^s is the set of SN edges. The attributes of an SN vertex $v^s \in V^s$ include its CPU capacity $\text{CPU}(v^s)$ and its location $\text{LOC}(v^s)$. The attribute of an SN edge $e^s \in E^s$ is its bandwidth capacity $\text{BW}(e^s)$. An SN path is a path that involves multiple edges in G^s.

B. Virtual Network Request

We define a VNR as an undirected graph $G^r = (V^r, E^r)$, where V^r is the set of VNR vertices and E^r is the set of VNR edges. The demands of a VNR vertex $v^r \in V^r$ are its CPU requirements $\text{CPU}(v^r)$, its preferred location $\text{LOC}(v^r)$, and the maximum allowed distance $D(v^r)$ from its preferred location to the location of the SN vertex it maps to. The popularly considered demand of a VNR edge $e^r \in E^r$ is its bandwidth requirement $\text{BW}(e^r)$.

C. Virtual Network Request Mapping

Given a VNR G^r, a feasible VNE is a mapping $\text{VNE}(\cdot)$ of VNR vertices to SN vertices and VNR edges to SN paths that satisfies 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 a unique SN vertex $\text{VNE}(v^r) \in V^s$,
2) no two VNR vertices $v_i^r \in V^r$ and $v_j^r \in V^r$ from the same VNR are mapped to the same SN vertex $v^s \in V^s$, and
3) for any VNR vertex $v^r \in V^r$:

$$\text{CPU}(v^r) \leq \text{CPU}(\text{VNE}(v^r))$$

and

$$\text{GEODIST}(\text{LOC}(v^r), \text{LOC}(\text{VNE}(v^r))) \leq D(v^r),$$

where $\text{GEODIST}(\cdot, \cdot)$ is the function that calculates the geographical distance between two 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 $\text{VNE}((v_i^r, v_j^r))$ from $\text{VNE}(v_i^r)$ to $\text{VNE}(v_j^r)$ in G^s, and
2) for any SN edge $e^s \in E^s$:

$$\sum\nolimits_{e^r \in E^r:\ e^s \in \text{VNE}(e^r)} \text{BW}(e^r) \leq \text{BW}(e^s).$$

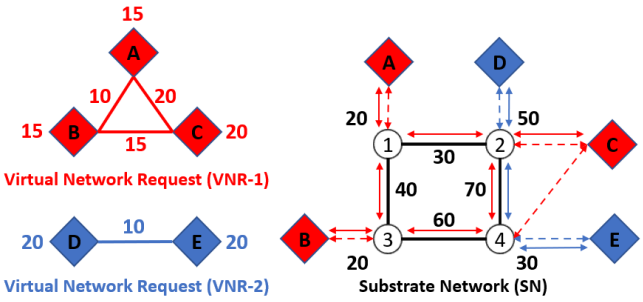

Fig. 1. An example embedding of two VNRs, VNR-1 and VNR-2, in an SN.

D. Objectives

We evaluate a feasible VNE mapping by its revenue and its cost. The revenue of a VNE mapping is defined as

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

and its cost is defined as

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

The VNE problem is to find a feasible VNE mapping, if one exists, that minimizes the cost of the mapping since the revenues of all feasible VNE mappings are identical.

E. Reformulating the VNE Problem to a Path-Coordination Problem

The VNE problem can be interpreted as a path-coordination problem after some reformulations [9].

We represent each VNR vertex $v^r \in V^r$ as a fictitious vertex $v^f \in V^f$ in a graphical representation of the SN. Each fictitious vertex $v^f \in V^f$ inherits all attributes of the VNR vertex v^r, such as $\text{CPU}(v^r)$, $\text{LOC}(v^r)$, and $D(v^r)$. Each v^f is connected via fictitious edges $(v^f, v^s) \in E^f$ to all SN vertices $v^s \in V^s$ that satisfy 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. This new graph G^m is called the augmented graph, and a VNE mapping is a set of paths on it. Its vertices are $V^s \cup V^f$, and its edges are $E^s \cup E^f$. Each VNR edge (v_i^r, v_j^r) that is mapped to an SN path can be traced on G^m with starting and ending fictitious edges (v_i^f, v_1^s) and (v_2^s, v_j^f), respectively, where $v_1^s, v_2^s \in V^s$. Such a path on 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.

Fig. 1 shows an example of successfully embedding two VNRs in 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 capacities, respectively. The non-negative numbers annotating the VNR edges and the SN edges represent their bandwidth requirements and 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 Fig. 1, the VNR vertex C can be mapped to either SN vertex 2 or 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 2-4.

In general, for any feasible VNE mapping of a given VNR in the SN, Eq. 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 have to be identical for all feasible 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 the bandwidth requirement $\text{BW}(e^r)$ multiplied by the length of the mapped SN path. Thus, minimizing the cost of a VNE mapping is the same as minimizing the sum of the lengths of the chosen paths on G^m. In turn, this corresponds to the sum-of-costs objective in the CBS framework.

IV. iVNE-CBS

The CBS framework was recently applied to the VNE problem [3]. The resulting solver, VNE-CBS, already outperforms popular baseline methods. In this section, we further improve VNE-CBS to iVNE-CBS using various enhancements to the high-level and low-level searches. We begin by describing the various kinds of conflicts that need to be resolved in the high-level search of VNE-CBS.

In the high-level search, every CT node N contains a VNE mapping $N.mapping$. The conflicts are constraints of the VNE problem that are violated by $N.mapping$. 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. To resolve it, two child CT nodes are created in the high-level search. One of these enforces the negative constraint $\overline{(v^r, v_1^s)}$ that stops v^r from being mapped to v_1^s, and the other enforces the negative constraint $\overline{(v^r, v_2^s)}$ that stops v^r from being mapped to v_2^s. 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 it, two child CT nodes are created in the high-level search. One of these enforces the negative constraint $\overline{(v_1^r, v^s)}$ that stops v_1^r from being mapped to v^s, and the other enforces the 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 a VNR edge e^r utilizes an SN edge e^s that does not have sufficient bandwidth capacity to accommodate $\text{BW}(e^r)$. To resolve it, we take all VNR edges in $\{e^r \in E^r : e^s \in \text{VNE}(e^r)\}$ and create a child CT node for each such VNR edge with a new negative constraint that

stops it from utilizing e^s. For each newly added negative constraint, the low-level search of VNE-CBS updates all paths that either map a VNR vertex to a prohibited SN vertex or use a prohibited SN edge in the mapping of a VNR edge.

In general, in the CBS framework, a CT node N also contains the cost of the VNE mapping $N.cost$ and the set of constraints imposed by the high-level search $N.constraints$. The geographical constraints are enforced while constructing the augmented graph G^m and, therefore, are not duplicated in $N.constraints$. A CT node N is a solution CT node if $N.mapping$ contains no conflicts.

A. Enhancements in the High-Level Search

In this subsection, we describe the two major enhancements incorporated by iVNE-CBS in its high-level search. The first one is referred to as disjoint splitting and the second one is referred to as bypassing conflicts. Both of these have been successfully used in the MAPF domain [19], [20].

1) Disjoint Splitting: Intuitively, the idea of disjoint splitting, first introduced in [19], 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 this overlap by using a conflict-resolution strategy based on constraint propagation. In this paper, we apply disjoint splitting to efficiently resolve vertex conflicts. It retains the completeness of iVNE-CBS.

Fig. 2 illustrates the application of disjoint splitting for resolving vertex conflicts in the CBS framework for the VNE problem. Here, VNR-3 contains three vertices and two edges that need to be embedded in the SN shown below it. The next panel shows a root CT node with a VNE mapping A-1-4-B and A-3-5-C. It contains a vertex conflict $(A, 1, A, 3)$, where the VNR vertex A is mapped to two SN vertices 1 and 3. VNE-CBS follows a non-disjoint splitting strategy for resolving this conflict. In particular, it splits the root 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 now prohibited by the newly added negative constraint. Similarly, at N_1, it replans the path A-3-5-C to A-2-5-C. It splits N_0 again 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, it splits N_1 to resolve the vertex conflict $(A, 1, A, 2)$ resulting in two child CT nodes N_{10} and N_{11}. Both N_{10} and N_{11} contain a feasible VNE mapping. However, 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 efforts in the CT subtrees.

To avoid duplication of search efforts, we use the disjoint splitting strategy, illustrated in the next panel. 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. It is clear that the complementary negative and positive constraints do not allow for any overlaps in the search efforts 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. The disjoint splitting strategy is different from brute force enumeration because of its implications on the low-level search. In the subtree with a

Algorithm 1 iVNE-CBS

1: Input: G^s, G^r, w
2: $G^m \leftarrow$ create 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 solution
16: $Conf \leftarrow$ first conflict found in $N_T.mapping$
17: $Cons \leftarrow$ (disjoint splitting) constraints for $Conf$
18: **for** $c \in Cons$ **do**
19: $N_C \leftarrow$ copy of N_T
20: Add c to $N_C.constraints$
21: Update low-level paths in $N_C.mapping$ to accommodate constraint c
22: **if** update 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
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"

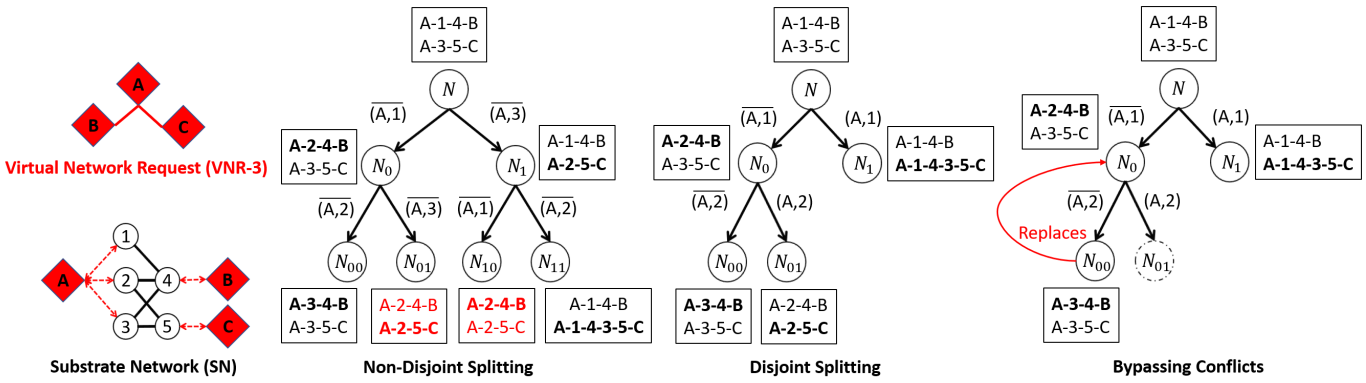

Fig. 2. An example of finding a VNE mapping for VNR-3 in the CBS framework with non-disjoint splitting, disjoint splitting, and bypassing conflicts. The first panel shows the VNR and the SN. The next panel shows the resolution of a vertex conflict in CT node N without using disjoint splitting. The two solutions marked in red indicate a duplication of search efforts. A path marked in bold indicates a path updated according to the constraints imposed by the high-level search. The next panel shows the resolution of the same vertex conflict using disjoint splitting. Here, redundant search efforts are avoided. The last panel shows how a CT node can replace its parent CT node if it has fewer conflicts and qualifies for expansion. Some details, such as the CPU and bandwidth resources, are omitted for simplicity.

negative constraint, the low-level search remaps the relevant VNR vertex to a new SN vertex. In the subtree with a positive constraint, the low-level search maps the VNR vertex to the specified SN vertex and ignores all other possibilities.

In the figure, we observe that disjoint splitting uses a smaller search tree. 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 $(A, 1)$. It further splits N_0 to resolve the vertex conflict $(A, 2, A, 3)$.

2) Bypassing Conflicts: Bypassing conflicts is a term that 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 [20]. We use the same strategy in the VNE domain as well. 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 whose number of conflicts is smaller than in N, we replace the paths in N with the paths in N_C and discard all generated child CT nodes of N. If there is no such child CT node N_C, we split the CT node N as usual. This strategy often reduces the number of CT node expansions to find a solution and, therefore, decreases the runtime of CBS. It can also be applied in focal search at the high level by modifying the criterion $N_C.cost = N.cost$ to $N_C.cost \leq w \cdot N.cost$ for choosing the child CT node [21].

In Fig. 2, the last panel shows an example of bypassing conflicts. iVNE-CBS with disjoint splitting finds a conflict $(A, 2, A, 3)$ at N_0 and splits on it. When it creates 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} is smaller than in N_0. It therefore replaces $N_0.mapping$ with $N_{00}.mapping$ and discards N_{00} and N_{01}. The new $N_0.mapping$ encodes a valid solution.

3) Pseudocode: Algorithm 1 shows the high-level search of iVNE-CBS. It is similar to the high-level search of VNE-CBS [3] but incorporates various enhancements shown in blue.

It takes three inputs: an SN graph G^s, a VNR graph G^r, and a suboptimality factor w. On Line 2, it uses G^s and G^r to create an augmented graph G^m, as described previously. On Line 3, it precomputes a table of true cost heuristics for the low-level search (explained later). On Lines 4-8, it initializes the root CT node N_R. On Line 6, it uses the low-level search to find a path from v_1^f to v_2^f on 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 (the initial VNE mapping). $N_R.cost$ is the cost of the mapping, and $N_R.num\_conf$ is the number of conflicts in it.

On Line 9, iVNE-CBS inserts N_R into priority queues *OPEN* and *FOCAL*. *OPEN* maintains a list of generated CT nodes, sorted by their costs. *FOCAL* maintains a list of CT nodes N in *OPEN* for which $N.cost \leq w \cdot OPEN.top().cost$, sorted by $N.num\_conf$. iVNE-CBS then expands CT nodes until either a feasible VNE mapping is found or *FOCAL* is empty. On Lines 11-13, it retrieves the CT 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 14-15, iVNE-CBS returns a feasible mapping if no conflicts exist in $N_T.mapping$. Otherwise, on Lines 16-17, it finds a conflict in $N_T.mapping$ and generates constraints *Cons* to resolve it. In resolving conflicts, iVNE-CBS prefers to resolve vertex conflicts before bandwidth capacity conflicts. If disjoint splitting is enabled, the resolution of vertex conflicts is done by generating the negative and positive constraints in two subtrees, as described in Fig. 2. In the subtree with positive constraints, the low-level search is adapted to satisfy them (explained later).

On Lines 18-20, iVNE-CBS generates a child CT node N_C

for each constraint $c \in$ *Cons* by making a copy of CT node N_T and adding the new constraint c to N_C.*constraints*. On Line 21, it updates N_C.*mapping* to accommodate the added constraint by invoking the low-level search to recompute the affected paths. If updating the paths is successful, on Lines 22-24, iVNE-CBS calculates the cost and number of conflicts for N_C. 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. On Lines 31-33, following the standard focal search procedure, iVNE-CBS inserts N_C into *OPEN* and into *FOCAL* if $N_C.cost$ is within w times the cost of the top node in *OPEN* from Line 11. On Lines 34-37, it updates *FOCAL* in the case that the cost of the top node in *OPEN* has increased as a result of previous operations. On Line 38, iVNE-CBS reports the absence of any solution.

B. Enhancements in the Low-Level Search

In this subsection, we describe the improved low-level search of iVNE-CBS. It is a best-first search that finds a shortest path from one fictitious start vertex v_1^f to another 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 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. True cost heuristics have been successfully used in the MAPF domain [11], [22] and are imported here for the VNE domain. They are particularly effective for 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. This often narrows the search space 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 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 a conflict with other paths [22].

In the VNE domain, iVNE-CBS uses a CAT for all kinds of conflicts, i.e., 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. (VNE-CBS minimizes the number of only type-2 vertex conflicts in its tie-breaking rule.)

Another implementation-level improvement in the low-level search of iVNE-CBS is more efficient 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 the open list resulting in downstream duplication of search efforts. In iVNE-CBS, we avoid this by checking the duplication of the child nodes. We record the generated child nodes with their f-values and allow the generation of a child node only if this node has not been previously generated. 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 lower 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.

Another new feature of the low-level search of iVNE-CBS is its ability to support positive constraints in $N.constraints$. This is done in the way it generates the neighbors of a node while searching for a path from v_1^f to v_2^f. 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 only returns 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.

In general, all fictitious vertices other than v_2^f are excluded as neighbors of any node since a path is not allowed to go through other fictitious vertices. All neighbors that are ruled out by the negative constraints are also excluded. Finally, the low-level search also rejects any path from v_1^f to v_2^f that maps two different VNR vertices to the same SN vertex since such a path would result in a type-2 vertex conflict.

V. EXPERIMENTS

In this section, we present experimental results comparing iVNE-CBS against VNE-CBS, D-ViNE, R-ViNE, G-SP, and RW-MaxMatch-SP. The last four algorithms are popular baseline methods for the core VNE problem. D-ViNE and R-ViNE heuristically retrieve a solution from an LP relaxation of an MIP model [9]. G-SP is a greedy algorithm that uses shortest path computations for mapping VNR edges [23]. RW-MaxMatch-SP is a node-ranking algorithm that uses the Markov Random Walk model for mapping VNR vertices to SN vertices based on the resources and topological attributes of vertices in a network. It maps VNR edges via shortest path computations [10]. We note that, compared to VNE-CBS, iVNE-CBS without disjoint splitting or bypassing conflicts incorporates only improvements in the low-level search.

We implemented iVNE-CBS and the baseline algorithms in C++. For iVNE-CBS and VNE-CBS, we experimented with $w = 1.0$, 1.5, and 2.0. Many configurations of the iVNE-CBS and VNE-CBS algorithms were studied. In the

experimental results, we refer to them using various self-explanatory suffixes. For example, VNE-CBS with $w = 1.0$ is labeled VNE-CBS-w1.0; iVNE-CBS with disjoint splitting (DS) is labeled iVNE-CBS-DS; and iVNE-CBS with disjoint splitting and bypassing conflicts (BC) is labeled iVNE-CBS-DS-BC. All experiments were run on an AWS machine with 8 CPUs and 16GB RAM. We set a timeout of 60 seconds for embedding a VNR in an SN.

We used Waxman graphs to generate VNE problem instances. Waxman graphs [24] are commonly used in the VNE literature to simulate communication networks. The SN topologies are randomly generated Waxman graphs with parameter values $\alpha = 0.3$ and $\beta = 0.1$. We generated 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 are real numbers generated uniformly at random from the interval $[50, 100]$.

We also used the Waxman method with $\alpha = 0.3$ and $\beta = 0.2$ to generate VNR topologies. We set the number of VNR vertices to be 10, 20, 30, 40, 50, 60, and 70, generating $1,000$ VNRs for each setting, with an average of 7.05, 18.84, 35.55, 57.39, 86.16, 122.31, and 164.94 edges, respectively. The VNR vertices were located in the same 100×100 grid space as the SN vertices. The maximum allowed Euclidean distance for the geographical constraints of VNR vertices was set to 15. The CPU requirements of the VNR vertices and the bandwidth requirements of the VNR edges are real numbers drawn uniformly at random from the interval $[0, 20]$ and $[0, 50]$, respectively.

We note that the SNs and VNRs used in our experiments are significantly larger than the SNs and VNRs used in previous works. While previous works use SNs with only about 100 vertices and 500 edges and VNRs with only about 10 vertices, here, we use SNs with 500 vertices and around $3,500$ edges and VNRs as large as with 70 vertices and around 164 edges.

A. Offline Experiments

In this subsection, we present the results of the experiments conducted in an offline setting. For each setting of the number of VNR vertices, we mapped each of the $1,000$ VNRs to each of the 3 SNs, i.e., we generated $3,000$ VNE instances for each setting. D-ViNE and R-ViNE showed very poor performance for solving our large-scale VNE instances. For VNE instances with 10 VNR vertices, D-ViNE and R-ViNE solved only 54 and 49 out of $3,000$ instances, respectively, with average runtimes of 58.276 and 58.440 seconds, respectively. They did not solve any of the larger instances. Therefore, they are not reported in the forthcoming results.

Fig. 3 shows the performances of VNE-CBS, iVNE-CBS, iVNE-CBS-DS, and iVNE-CBS-DS-BC for different w. The major comparison metrics include the success rate, the average cost of the solution, the average runtime, and the average number of CT node expansions required to find a solution. The success rate is the fraction of VNE instances that are successfully solved. No data point is reported for an algorithm if it failed to solve any VNE instance. The other metrics are averaged over the VNE instances that are successfully solved for each setting of the number of VNR vertices.

In general, the success rates of iVNE-CBS algorithms are much better than that of VNE-CBS and increase with increasing w. Moreover, disjoint splitting also increases the success rate and is even better when combined with bypassing conflicts. In fact, iVNE-CBS-DS-w2.0 significantly outperforms VNE-CBS and achieves a 98.46% success rate for VNE instances with 70 VNR vertices. Bypassing conflicts allows us to solve 46 more VNE instances in this realm, increasing the success rate to 99.66%. The average costs of the solutions produced by all algorithms are similar. This is because both VNE-CBS and iVNE-CBS algorithms with $w = 1.0$ produce optimal solutions. For larger values of w, iVNE-CBS algorithms are guaranteed to produce w-suboptimal solutions but in practice produce close-to-optimal solutions. The average runtimes of iVNE-CBS algorithms are also better than that of VNE-CBS and improve with increasing w. Moreover, disjoint splitting also improves the average runtime and is sometimes benefited when combined with bypassing conflicts. In fact, iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 significantly outperform iVNE-CBS-w2.0, indicating the benefits of disjoint splitting and bypassing conflicts. The same trends are also observed for the average number of expanded CT nodes required to find a solution.

From Fig. 3, we draw iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 as the two best CBS-based algorithms for comparison against other baseline methods. We also include VNE-CBS with the same w in the comparison for evaluating the enhancements in iVNE-CBS. Fig. 4 shows the comparison of the chosen algorithms. The iVNE-CBS algorithms have the best success rates, best average costs, as well as the best average runtimes.

The average runtime depends on: (a) the average number of expanded CT nodes and (b) the average number of low-level nodes expanded per CT node. Fig. 3 shows that the high-level enhancements in iVNE-CBS significantly reduce (a), and Fig. 5 shows a comparison on (b). We observe that the low-level enhancements in iVNE-CBS reduce (b) significantly, contributing to its superior performance over VNE-CBS even without high-level enhancements.

B. Online Experiments

In this subsection, we present the results of the experiments conducted in an online setting. Here, VNRs arrive dynamically at different timesteps, and each successfully embedded (accepted) VNR holds the SN resources allocated to it until it departs at the end of its lifetime. VNRs arrive according to a Poisson process at an average rate of 4 VNRs per 100 timesteps. The lifetime of each VNR is drawn from an exponential distribution with an average of $1,000$ timesteps. When mapping a new VNR, no algorithm is allowed to reconfigure the mapping of previously embedded VNRs. If an algorithm fails to map a VNR within 60 seconds, it rejects the VNR and tries the next one. We choose $w = 2.0$ since it yields the best performance in the offline experiments.

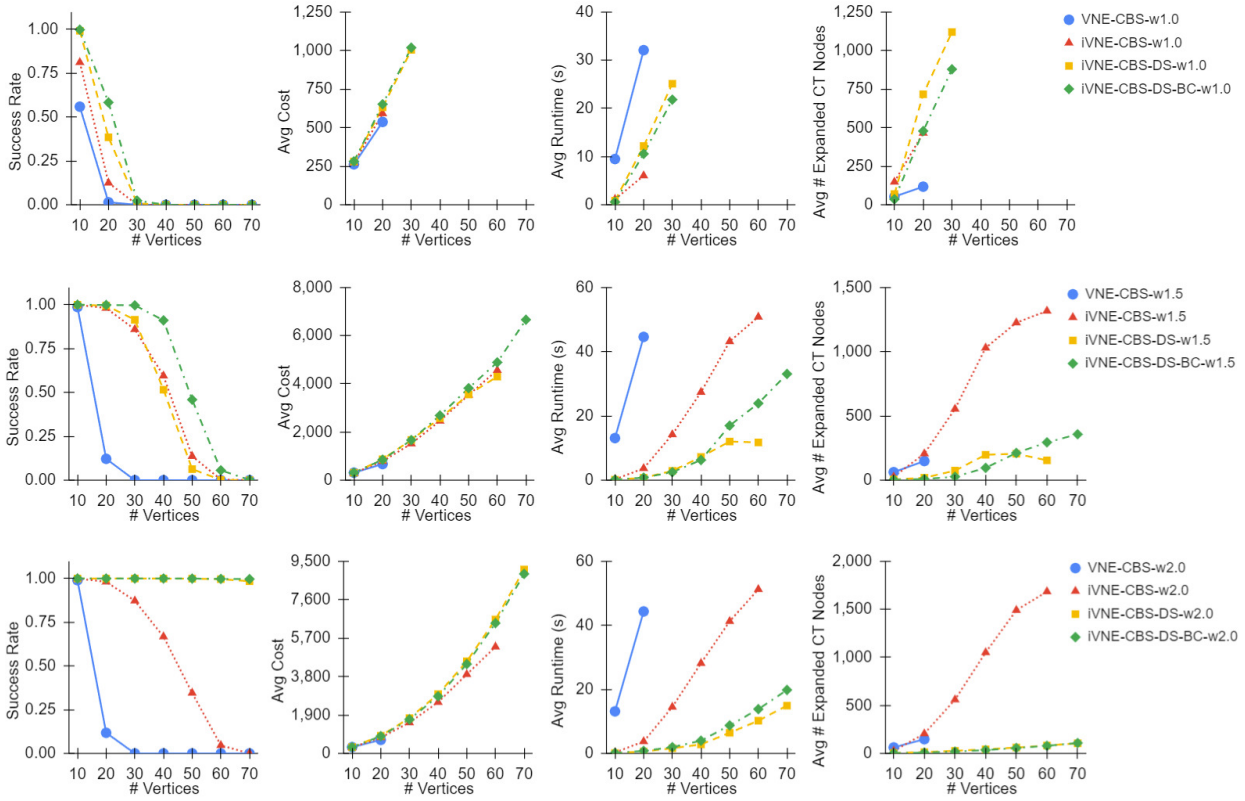

Fig. 3. Offline Experiment: comparing VNE-CBS and variants of iVNE-CBS with $w = 1.0$, 1.5, and 2.0.

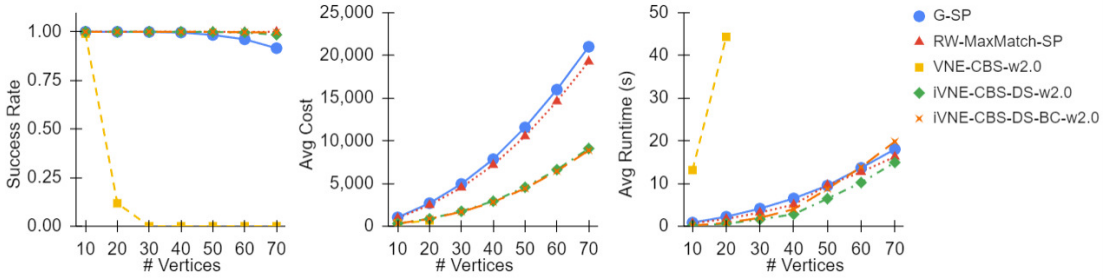

Fig. 4. Offline Experiment: comparing iVNE-CBS-DS-w2.0 and iVNE-CBS-DS-BC-w2.0 against the baseline methods.

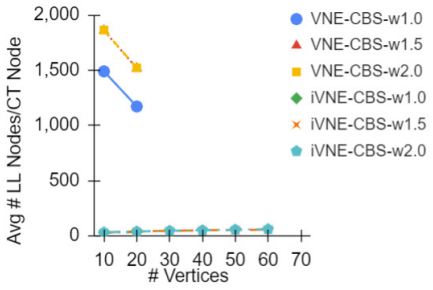

Fig. 5. Offline Experiment: comparing the average numbers of low-level nodes expanded per CT node for iVNE-CBS and VNE-CBS.

The experiments utilize 3 runs corresponding to the 3 SNs, each on the same $1,000$ VNRs with a given number of VNR vertices. Fig. 6 presents the results on the metrics of cost/revenue, acceptance ratio, and total revenue, averaged over the 3 runs. The cost/revenue is the cost divided by the revenue for each accepted VNR. It is averaged over all accepted VNRs across the 3 runs. On this metric, the iVNE-CBS algorithms outperform all other algorithms, indicating that they allocate SN resources to accepted VNRs more efficiently. The acceptance ratio is the fraction of accepted VNRs. On this metric too, the iVNE-CBS algorithms outperform VNE-CBS and other baseline algorithms due to the efficient allocation of SN resources to previously accepted VNRs. The total revenue is the sum of the revenue of the accepted VNRs. On this metric, the iVNE-CBS algorithms outperform VNE-CBS and other baseline algorithms before dropping slightly for 70 VNR vertices. The superior performance of iVNE-CBS algorithms

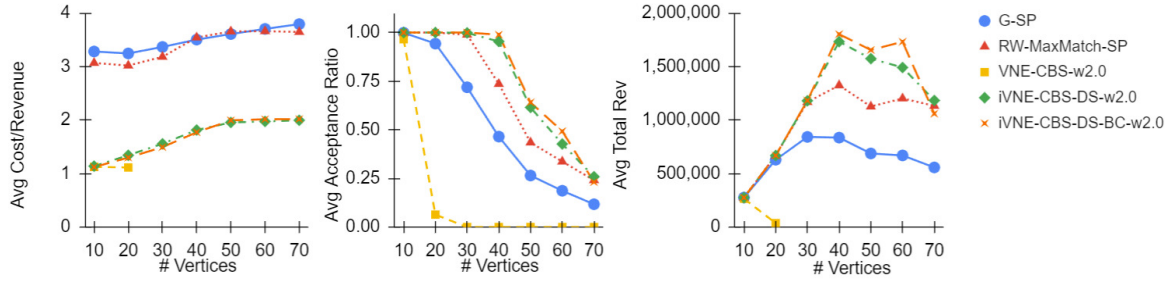

Fig. 6. Online Experiment: comparing iVNE-CBS and its variants with $w = 2.0$ against other competing algorithms.

is because of their higher acceptance ratios.

VI. Conclusions

In this paper, we presented iVNE-CBS, a new solver for the VNE problem that is foundational to network virtualization and slicing. iVNE-CBS improves on VNE-CBS using various enhancements in the high-level and low-level searches, such as disjoint splitting, bypassing conflicts, and true cost heuristics. These enhancements are inspired by similar enhancements in the MAPF domain. iVNE-CBS significantly outperforms VNE-CBS and other popular VNE algorithms both in the offline and online settings and in terms of both runtime and solution quality. iVNE-CBS scales well to solve large VNE instances, much beyond the scale of the instances previously studied in the VNE literature. Its success is indicative of our ability to draw power from MAPF technologies and cross-fertilize ideas between the two domains.

VII. 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 author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government.

References

[1] 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.

[2] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[3] Y. Zheng, S. Ravi, E. Kline, S. Koenig, and T. K. S. Kumar, "Conflict-based search for the virtual network embedding problem," in *International Conference on Automated Planning and Scheduling*, 2022, pp. 423–433.

[4] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Symposium on Combinatorial Search*, 2019, pp. 151–159.

[5] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, 2012.

[6] D. G. Andersen, "Theoretical approaches to node assignment," Ph.D. dissertation, Carnegie Mellon University, 2002.

[7] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[8] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A survey of embedding algorithm for virtual network embedding," *China Communications*, vol. 16, no. 12, pp. 1–33, 2019.

[9] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Joint Conference of the IEEE Computer and Communications Societies*, 2009, pp. 783–791.

[10] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *Computer Communication Review*, vol. 41, no. 2, pp. 38–47, 2011.

[11] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[12] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *AAAI Conference on Artificial Intelligence*, 2013, pp. 1443–1449.

[13] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding with payload transfers and the package-exchange robot-routing problem," in *AAAI Conference on Artificial Intelligence*, 2016, pp. 3166–3173.

[14] D. Silver, "Cooperative pathfinding," in *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2005, pp. 117–122.

[15] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–20, 2008.

[16] S. Choudhury, K. Solovey, M. J. Kochenderfer, and M. Pavone, "Efficient large-scale multi-drone delivery using transit networks," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 4543–4550.

[17] O. Gordon, Y. Filmus, and O. Salzman, "Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds," in *International Symposium on Combinatorial Search*, vol. 12, 2021, pp. 64–72.

[18] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *International Symposium on Combinatorial Search*, 2014, pp. 19–27.

[19] J. Li, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Disjoint splitting for multi-agent path finding with conflict-based search," in *International Conference on Automated Planning and Scheduling*, 2019, pp. 279–283.

[20] E. Boyarski, A. Felner, G. Sharon, and R. Stern, "Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding," in *International Conference on Automated Planning and Scheduling*, 2015, pp. 47–51.

[21] J. Li, W. Ruml, and S. Koenig, "EECBS: Bounded-suboptimal search for multi-agent path finding," in *AAAI Conference on Artificial Intelligence*, 2021, pp. 12 353–12 362.

[22] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *AAAI Conference on Artificial Intelligence*, 2010.

[23] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Joint Conference of the IEEE Computer and Communications Societies*, 2006, pp. 1–12.

[24] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.