Priority-Based Search for the Virtual Network Embedding Problem

Yi Zheng[1], **Hang Ma**[2], **Sven Koenig**[1], **Erik Kline**[1], **T. K. Satish Kumar**[1]

[1] University of Southern California
[2] Simon Fraser University
yzheng63@usc.edu, hangma@sfu.ca, skoenig@usc.edu, kline@isi.edu, tkskwork@gmail.com

Abstract

The Virtual Network Embedding (VNE) problem is a constrained optimization problem. It arises in the context of allocating resources on heterogeneous physical networks to provide end-to-end computing services. In this paper, we introduce a new solver, called VNE-PBS, that uses priority-based search (PBS) for solving the VNE problem. VNE-PBS uses a prioritized heuristic search algorithm that explores the space of all possible priority orderings using a systematic depth-first search. The solver is inspired by the success of PBS for the Multi-Agent Path Finding (MAPF) problem and the similarities between the VNE and MAPF problems. We show that VNE-PBS significantly outperforms competing methods on various benchmark instances for both the offline and online versions of the VNE problem.

Introduction

Network virtualization is an enabling technology that aims to overcome the Internet ossification problem, a reference to the resistance of the current Internet to architectural changes (Chowdhury, Rahman, and Boutaba 2009). Through network virtualization, significant investment to homogenize the physical infrastructure can be avoided. Instead, a proper allocation of the resources across heterogeneous physical networks can help service providers provision end-to-end services to the customers. They can do so by leasing the shared network resources from infrastructure providers (Feamster, Gao, and Rexford 2007). Network virtualization also facilitates increased security and manageability (Fischer et al. 2013).

The physical infrastructure managed by infrastructure providers can be conceptualized as an undirected graph, often referred to as the Substrate Network (SN). The capacities of an SN include its CPU capacity (i.e., the compute power on its vertices) and its bandwidth capacity (i.e., the communication capacity on its edges). A request for network resources can also be conceptualized as an undirected graph, often referred to as a Virtual Network Request (VNR). In a VNR, each vertex is annotated with a CPU requirement, and each edge is annotated with a bandwidth requirement.

Network virtualization hinges on our ability to manage the SN resources (i.e., the compute power on the SN vertices and the communication capacity on the SN edges) efficiently and effectively. This resource management problem is formalized as the Virtual Network Embedding (VNE) problem. In the VNE problem, each VNR vertex is required to be mapped to an SN vertex (the vertex/node mapping), and each VNR edge is required to be mapped to an SN path (the edge/link mapping). The mapping must satisfy the CPU and bandwidth capacity constraints on the SN vertices and edges, respectively. It may also have to satisfy geolocation constraints on the individual VNR vertices. The VNE problem and its many variants are NP-hard (Yu et al. 2008).

Recently, the Conflict-Based Search (CBS) algorithm, imported from the Multi-Agent Path Finding (MAPF) literature, has been successfully applied to the VNE problem (Zheng et al. 2022). The resulting VNE-CBS algorithm exploits the combinatorial similarities between the VNE and MAPF problems. In the MAPF problem, we are given a team of agents and an undirected graph. Each agent has a distinct start vertex and a distinct goal vertex. It has to move from its start vertex to its goal vertex while avoiding collisions (also called conflicts) with the other agents. A conflict happens when two agents stay at the same vertex or traverse the same edge in opposite directions at the same time. Each action, such as moving to a neighboring vertex or staying at the current vertex, has a cost. A common objective is to minimize the sum of the travel costs of the agents. Solving the MAPF problem optimally for this objective is NP-hard (Yu et al. 2008).

There are many similarities between the MAPF and VNE problems. The VNE problem can be construed as a constrained path-coordination problem (Chowdhury, Rahman, and Boutaba 2009). This makes it amenable to MAPF algorithms (Zheng et al. 2022). Similar to CBS for MAPF, VNE-CBS 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 element under the constraints imposed by the high-level search node. On the one hand, VNE-CBS exploits the similarities between the VNE and MAPF problems. On the other hand, it also successfully addresses the subtle differences and unique challenges in applying the CBS framework to the VNE problem. VNE-CBS is complete and optimal.

Priority-Based Search (PBS) is another successful MAPF algorithm. Similar to CBS, it is also a two-level search algorithm. However, it conducts its high-level search on a priority tree, different from the conflict resolution tree used in CBS. It uses a depth-first search instead of the best-first search used in CBS. In the MAPF problem, the low-level search of PBS finds a path for an agent that does not conflict with the paths of the agents that have been assigned higher priorities in the high-level search node. It conducts its low-level search by treating higher-priority paths as constraints.

In this paper, we present a novel adaptation of the PBS algorithm for solving the VNE problem efficiently and effectively. Our algorithm, called VNE-PBS, conducts a two-level search, where the high-level search resolves conflicts on a priority tree and the low-level search allocates SN resources to VNR vertices and edges while avoiding conflicts with high-priority allocations. VNE-PBS intelligently uses priorities for mapping both VNR vertices and edges simultaneously to SN vertices and paths, respectively. It also explores the space of all possible priority orderings using a depth-first search. We empirically show that VNE-PBS significantly outperforms competing algorithms on various benchmark instances in both offline and online settings.

Background

In this section, we present the background literature relevant to our work. In doing so, we also introduce important concepts and definitions that will be useful later.

Virtual Network Embedding

The VNE problem is essentially a constrained resource allocation problem that maps a VNR to an SN.

In the VNE problem, an SN is an undirected graph $G^s = (V^s, E^s, A^s_V, A^s_E)$, where V^s is the set of SN vertices, E^s is the set of SN edges, A^s_V is a mapping from SN vertices to their attributes, and A^s_E is a mapping from SN edges to their attributes. For $v^s \in V^s$, $A^s_V(v^s)$ includes its CPU capacity $\textsc{cpu}(v^s)$ and its location $\textsc{loc}(v^s)$. For $e^s \in E^s$, $A^s_E(e^s)$ includes its bandwidth capacity $\textsc{bw}(e^s)$. An SN path is a path in G^s. A VNR is also an undirected graph $G^r = (V^r, E^r, C^r_V, C^r_E)$, where V^r is the set of VNR vertices, E^r is the set of VNR edges, C^r_V is a mapping from VNR vertices to their demands, and C^r_E is a mapping from VNR edges to their demands. For $v^r \in V^r$, $C^r_V(v^r)$ includes its CPU requirement $\textsc{cpu}(v^r)$ and its geolocation requirement specified as a maximum distance $D(v^r)$ between its desired location $\textsc{loc}(v^r)$ and the location of the SN vertex it is mapped to. For $e^r \in E^r$, $C^r_E(e^r)$ includes its bandwidth requirement $\textsc{bw}(e^r)$.

Given an SN G^s and a VNR G^r, a solution to the VNE problem is a feasible mapping $\textsc{vne}(\cdot)$ of VNR vertices to SN vertices and VNR edges to SN paths. The mapping must satisfy a number of constraints. The commonly used constraints are the following:

- Each VNR vertex $v^r \in V^r$ is mapped to a unique SN vertex $\textsc{vne}(v^r) \in V^s$,
- No two VNR vertices $v^r_i \in V^r$ and $v^r_j \in V^r$ from the same VNR are mapped to the same SN vertex $v^s \in V^s$
- Each VNR vertex $v^r \in V^r$ is mapped to exactly one SN vertex such that $\textsc{cpu}(v^r) \leq \textsc{cpu}(\textsc{vne}(v^r))$ and $\textsc{geodist}(\textsc{loc}(v^r), \textsc{loc}(\textsc{vne}(v^r))) \leq D(v^r)$, where $\textsc{geodist}(\cdot,\cdot)$ is the geographical distance function between two locations.
- Each VNR edge $e^r = (v^r_i, v^r_j) \in E^r$ is mapped to an SN path $\textsc{vne}((v^r_i, v^r_j))$ from $\textsc{vne}(v^r_i)$ to $\textsc{vne}(v^r_j)$ in G^s, such that, for any SN edge $e^s \in E^s$, the sum of the bandwidth requirements of the VNR edges that utilize it does not exceed its bandwidth capacity, that is, $\sum_{e^r \in E^r:\, e^s \in \textsc{vne}(e^r)} \textsc{bw}(e^r) \leq \textsc{bw}(e^s)$.

Several metrics are popular for evaluating VNE mappings. One of them is the revenue. It is the sum of the resources requested by and successfully meted out to the VNR. It is 0 if the VNR cannot be embedded in the SN. Otherwise, it is given by

$$\sum\nolimits_{v^r \in V^r} \textsc{cpu}(v^r) + \sum\nolimits_{e^r \in E^r} \textsc{bw}(e^r). \qquad (1)$$

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 in the SN. Otherwise, it is given by

$$\sum\nolimits_{v^r \in V^r} \textsc{cpu}(v^r) + \sum\nolimits_{e^r \in E^r} \sum\nolimits_{e^s \in \textsc{vne}(e^r)} \textsc{bw}(e^r). \qquad (2)$$

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. 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.

The VNE problem and its many variants are NP-hard (Yu et al. 2008). However, many practical solvers have been proposed. The VNE problem with geolocation requirements can be formulated as a Mixed Integer Linear Programming (MILP) problem. This MILP problem can then be relaxed to a Linear Programming (LP) problem that can be solved in polynomial time. Two solvers, D-ViNE and R-ViNE (Chowdhury, Rahman, and Boutaba 2009), heuristically retrieve a solution of the VNE problem from the fractional solution of the relaxed LP problem via a deterministic and a randomized rounding technique, respectively. These two solvers are often used as the standard baseline solvers for comparatively evaluating new VNE solvers.

G-SP and G-MCF (Yu et al. 2008) are two solvers that first greedily map VNR vertices to SN vertices and then use shortest-path or multi-commodity flow computations to map VNR edges to SN paths. Some VNE solvers use node ranking to improve the greedy mapping of VNR vertices to SN vertices. For example, drawing inspiration from Google's Page Rank algorithm, RW-MaxMatch-SP sorts the VNR and SN vertices and maps them according to their ranks (Cheng et al. 2011). Then, it uses a shortest-path algorithm to map the VNR edges to SN paths. Fischer et al. (2013) and Cao et al. (2019) provide detailed surveys of the VNE problem, its variants, and a classification of their solvers.

Recently, Zheng et al. (2022) proposed a CBS algorithm for solving the VNE problem. The solver, VNE-CBS, is inspired by the success of the CBS algorithm for the MAPF problem and the fact that the VNE problem can be converted to a path-coordination problem, as proposed in (Chowdhury, Rahman, and Boutaba 2009). VNE-CBS is a two-level heuristic search algorithm. The high-level search is a best-first search that resolves any constraint violations (also called conflicts) in resource allocation for the VNR vertices and edges. In resolving conflicts, each high-level search node imposes constraints on the allocation of SN resources. The low-level search adheres to these constraints and maps individual VNR vertices and edges to SN vertices and paths, respectively. VNE-CBS demonstrates the benefit of importing technologies from the MAPF literature.

Multi-Agent Path Finding

In the MAPF problem, the task is to find paths for a set of k agents $\{a_1, ..., a_k\}$ on a graph $G = (V, E)$, where each agent a_j has a distinct start vertex $s^j \in V$ and a distinct goal vertex $g^j \in V$. Time is discretized into timesteps, and each agent can either move to a neighboring vertex or wait at its current vertex at any timestep t. Each action has a cost. A path of an agent is a sequence of move and wait actions that lead the agent from its start vertex to its goal vertex. A conflict arises when two agents visit the same vertex or traverse the same edge at the same time. A solution to a MAPF problem is a set of paths, one for each agent, without any conflicts. A popular objective is to minimize the sum of the travel costs of all agents. The MAPF problem (Stern et al. 2019) arises in many real-world applications, including video games (Silver 2005), automated warehousing (Wurman, D'Andrea, and Mountz 2008), and multi-drone delivery (Choudhury et al. 2020).

Priority-Based Search in Multi-Agent Path Finding

Prioritized MAPF algorithms (Silver 2005; Sturtevant and Buro 2006) are among the most efficient algorithms for solving MAPF problems. They are based on a simple *prioritized planning scheme*: Each agent is given a unique priority and computes, in priority order, a minimum-cost path from its start vertex to its goal vertex that does not conflict with the already planned paths of all higher-priority agents (Erdmann and Lozano-Pérez 1987). This category of algorithms excels in terms of runtime and is often used in MAPF solvers (Velagapudi, Sycara, and Scerri 2010; Wang and Botea 2011; Cáp, Vokrínek, and Kleiner 2015). However, these solvers use a predefined total priority ordering on the agents that can result in solutions of bad quality or even fail to find any solutions for solvable MAPF instances, where different total priority orderings could have resulted in better solution qualities or avoided failure. An efficient MAPF algorithm, called Priority-Based Search (PBS), was proposed to address this issue (Ma et al. 2019).

PBS generalizes prioritized planning from planning with a predefined total priority ordering on the agents to planning with all possible total priority orderings. It explores the space of all total priority orderings lazily using a systematic depth-first search. PBS is a two-level search algorithm. On the high level, it performs a depth-first search to construct a priority ordering dynamically and thus builds a priority tree. Each high-level node N contains a priority ordering given by a set $\prec_N$ of priority relationships between agents. Each high-level node also has a set of paths, one for each agent, $N.paths$ that respect the prioritized planning scheme for the priority ordering $\prec_N$. The cost of a high-level node N is the sum of the travel costs of the paths in $N.paths$. The root high-level node contains an empty set of priority relationships and a set of shortest paths that may contain conflicts. When PBS expands a high-level node N, it first checks for conflicts in $N.paths$. If there are none, then the high-level node is a goal high-level node and the solution is its set of paths. If there is a conflict between the paths of agent a_i and agent a_j, then PBS resolves it by splitting N into two child high-level nodes N_1 and N_2. For child high-level nodes N_1 and N_2, it introduces an additional priority relationship $j \prec i$ (agent a_j has a higher priority than agent a_i) and $i \prec j$ (agent a_i has a higher priority than agent a_j) to $\prec_{N_1}$ and $\prec_{N_2}$, respectively. In child high-level node N_1, PBS updates the paths of agent a_i and all agents a_k with $i \prec_{N_1} k$. The processing of child high-level node N_2 is symmetric. Each update is done in the low-level search of PBS and results in a minimum-cost path for each agent that does not conflict with the paths of all higher-priority agents. The low-level search for updating the path of an agent is a single-agent path finding procedure that expands only those low-level nodes that do not result in conflicts with the paths of the higher-priority agents.

Reformulating the VNE Problem as a Path-Coordination Problem

In this section, we explain how the VNE problem can be converted to a path-coordination problem on an artificially created *augmented* SN, as proposed in (Chowdhury, Rahman, and Boutaba 2009).

We first create a fictitious vertex $v^f \in V^f$, where V^f is the set of fictitious vertices, for each VNR vertex $v^r \in V^r$. v^f inherits all attributes of v^r, including $\text{CPU}(v^r)$, $\text{LOC}(v^r)$, and $D(v^r)$. We then create fictitious edges E^f to connect the fictitious vertices to the SN vertices. Each fictitious vertex v^f is connected to all SN vertices that satisfy both its CPU requirement and the geolocation constraint. In other words, each v^f is connected via fictitious edges (v^f, v^s) to a set of SN vertices $\{v^s \in V^s : \text{CPU}(v^f) \leq \text{CPU}(v^s)$ and $\text{GEODIST}(\text{LOC}(v^f), \text{LOC}(v^s)) \leq D(v^f)\}$. The bandwidth of a fictitious edge is set to infinity. The augmented SN is the union of the original SN, the fictitious vertices, and the fictitious edges. We denote it by $G^m = (V^m, E^m)$, where $V^m = V^s \cup V^f$ and $E^m = E^s \cup E^f$.

On the augmented SN, the VNE mapping of a VNR edge (v_i^r, v_j^r) can be identified by a path $\langle v_i^f, v_1^s, \ldots, v_2^s, v_j^f \rangle$ on G^m, where v_i^f and v_j^f are the fictitious vertices corresponding to v_i^r and v_j^r, respectively, and $(v_i^f, v_1^s), (v_2^s, v_j^f) \in E^f$. 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. This reformulation bears a resemblance to the MAPF problem, where a path is required for each agent from its start vertex

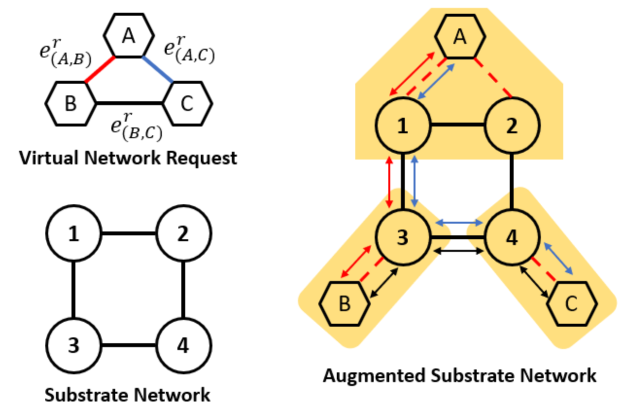

Figure 1: An example of a reformulated VNE problem. The VNR vertices A, B, and C are added as fictitious vertices in the augmented SN. Their connections to qualifying SN vertices that satisfy the CPU and geolocation constraints are indicated by the yellow areas. Fictitious edges are shown as red dashed lines. Each VNR edge is implemented as a path on the augmented SN, bearing the same color. For instance, the blue path A-1-3-4-C represents the mapping of VNR vertices A and C to SN vertices 1 and 4, respectively, and the mapping of the blue VNR edge $e^r_{(A,C)}$ to the SN path 1-3-4.

to its goal vertex. Each VNR edge roughly corresponds to an agent, although the MAPF problem has a temporal dimension that is not present in the VNE problem. Figure 1 shows an example.

In general, for any feasible VNE mapping that maps a given VNR into the given SN, Eq. 2 indicates that the cost of the VNE mapping depends on the resources allocated to the VNR. The CPU resources allocated to the VNR vertices are the same 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 equal to its bandwidth requirement $\text{BW}(e^r)$ multiplied by the length of the SN path to which it is mapped. Thus, minimizing the cost of a VNE mapping is equivalent to minimizing the sum of the lengths of the chosen paths on G^m. This corresponds to the sum of the travel costs objective in the MAPF problem, furthering the combinatorial similarity between the VNE and MAPF problems.

Conflict Resolution and Prioritization

The VNE problem is a path-coordination problem similar to the MAPF problem. We now identify *conflicts* that can arise for resources allocated by a VNE mapping. A feasible VNE mapping of a VNR to an SN must satisfy multiple constraints. The violations of these constraints characterize the conflicts (i.e., a feasible VNE mapping is conflict-free). For the VNR edge (v^r_i, v^r_j) implemented as the G^m path $\langle v^f_i, v^s_1, \dots, v^s_2, v^f_j \rangle$, v^s_1 and v^s_2 represent the mapping of the VNR vertices v^r_i and v^r_j, respectively, and the SN path $\langle v^s_1, \dots, v^s_2 \rangle$ represents the mapping of the VNR edge (v^r_i, v^r_j).

The first kind of constraints on a VNE mapping is that each VNR vertex must be mapped to a distinct SN vertex. Therefore, we have the following kinds of conflicts:

- A *type-1 vertex conflict* (v^r, v^s_1, v^r, v^s_2) arises when the G^m paths for two VNR edges e^r_1 and e^r_2 map the same VNR vertex v^r to two different SN vertices v^s_1 and v^s_2, respectively.
- A *type-2 vertex conflict* (v^r_1, v^s, v^r_2, v^s) arises when the G^m paths for two VNR edges e^r_1 and e^r_2 map two different VNR vertices v^r_1 and v^r_2, respectively, to the same SN vertex v^s. It also arises when the G^m path for a VNR edge $e^r = (v^r_1, v^r_2)$ maps two different VNR vertices v^r_1 and v^r_2 to the same SN vertex v^s.

The second kind of constraints on a VNE mapping is on the CPU and geolocation attributes of the SN vertices that a VNR vertex can be mapped to. Such constraints are incorporated by design in the construction of G^m. Therefore, no conflicts arise in this regard.

The third kind of constraints on a VNE mapping is that the sum of the bandwidth requirements of the VNR edges that utilize an SN edge cannot exceed its bandwidth capacity. Therefore, we have the *bandwidth capacity conflict* $E^c = \{e^r \in E^r : \sum_{e^r:e^s \in \text{VNE}(e^r)} \text{BW}(e^r) > \text{BW}(e^s)\}$ that arises when an SN edge e^s does not have sufficient bandwidth capacity to accommodate all VNR edges that utilize it.

Priorities can be used to obtain a feasible mapping. In previous works on the VNE problem, priorities have been used in greedy algorithms, such as in G-SP, G-MCF (Zhu and Ammar 2006), and RW-MaxMatch-SP (Cheng et al. 2011). However, their use of priorities is limited: Their first objective is to map the VNR vertices to SN vertices greedily. The VNR edges are later mapped to SN paths only in accordance with the vertex mappings. In VNE-PBS, we use priorities in more sophisticated ways. First, the priorities are associated with the G^m paths, which simultaneously map the VNR vertices and edges to SN vertices and paths, respectively. Second, all possible priority orderings are explored. VNE-PBS thus combines the benefits of priorities with a two-level search procedure similar to PBS for the MAPF problem.

Given two VNR edges $e^r_1 = (v^r_i, v^r_j)$ and $e^r_2 = (v^r_k, v^r_l)$, we use $e^r_1 \prec e^r_2$ to denote that the path from v^f_i to v^f_j on G^m is found with a higher priority than the path from v^f_k to v^f_l. When finding any G^m path, VNE-PBS constrains that path to avoid all vertex conflicts and bandwidth capacity conflicts with the higher-priority G^m paths. There can be conflicts between G^m paths with no priority relationship between them. Any such conflict is resolved by adding a priority relationship between every pair of G^m paths participating in the conflict.

To resolve a type-1 vertex conflict (v^r, v^s_1, v^r, v^s_2) between two G^m paths found for e^r_1 and e^r_2, we create two child high-level nodes N_1 and N_2. For N_1, we add a priority relationship $e^r_1 \prec e^r_2$ to $\prec_{N_1}$ and replan the path for e^r_2 and all paths with a lower priority than e^r_2. The new path for e^r_2

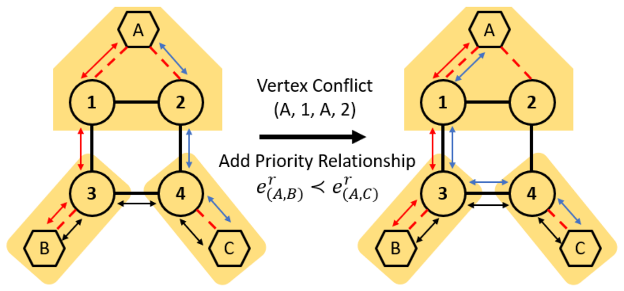

Figure 2: An example of resolving a type-2 vertex conflict. The type-2 vertex conflict $(A, 1, A, 2)$ between the G^m paths implemented for $e^r_{(A,B)}$ and $e^r_{(A,C)}$ is resolved by adding the priority relationship $e^r_{(A,B)} \prec e^r_{(A,C)}$ in one of the child high-level nodes. Therefore, the G^m path from A to C (blue) is changed to A-1-3-4-C so that A is mapped to 1 to agree with the path from A to B (red).

must map the VNR vertex v^r to v^s_1 instead of v^s_2 since it cannot conflict with any higher-priority path, including that for e^r_1. The procedure is symmetric for N_2. To resolve a type-2 vertex conflict (v^r_1, v^s, v^r_2, v^s) between two G^m paths found for e^r_1 and e^r_2, we create two child high-level nodes N_1 and N_2. As before, for N_1, we add a priority relationship $e^r_1 \prec e^r_2$ to $\prec_{N_1}$ and replan the path for e^r_2 and all paths with a lower priority than e^r_2. The new path for e^r_2 must map the VNR vertex v^r_2 to an SN vertex different from v^s since the new path cannot conflict with any higher-priority path. The procedure is symmetric for N_2. Figure 2 shows an example of resolving a type-2 vertex conflict.

As described before, a bandwidth capacity conflict $E^c = \{e^r \in E^r : \sum_{e^r:e^s\in \text{VNE}(e^r)} \text{BW}(e^r) > \text{BW}(e^s)\}$, arises when an SN edge e^s does not have sufficient bandwidth capacity to accommodate all VNR edges that utilize it. Such a conflict is first recognized and triggered by a VNR edge e^r_1 that cannot be accommodated by e^s when the G^m path for e^r_1 wants to utilize e^s. To resolve the conflict, we create two child high-level nodes N_1 and N_2 for each VNR edge $e^c \in E^c \setminus \{e^r_1\}$. For N_1, we add a priority relationship $e^c \prec e^r_1$ to $\prec_{N_1}$ and replan the path for e^r_1. The new path for e^r_1 must not utilize e^s. For N_2, we add a priority relationship $e^r_1 \prec e^c$ and follow a symmetric procedure.

Our conflict-resolution methods address all kinds of conflicts and explore all possible combinations of priorities for implementing the VNR edges as G^m paths.

VNE-PBS

In this section, we describe the algorithmic aspects of VNE-PBS, see Algorithm 1.

High-Level Search

Algorithm 1 shows the high-level search of VNE-PBS. It takes two inputs: the SN graph G^s and the VNR graph G^r. On Line 2, it uses G^s and G^r to create the augmented SN G^m, as described previously. On Line 3, it precomputes a table of true cost heuristic values for guiding the low-level search. The true cost heuristic values are the shortest hop distances from all fictitious vertices to all other vertices in the augmented SN.

Algorithm 1: VNE-PBS

```
1:  Input: $G^s$, $G^r$
2:  $G^m \leftarrow$ create augmented SN for $G^s$ and $G^r$
3:  Precompute the true cost heuristic table $h\_table$
4:  $N_R \leftarrow$ empty high-level node
5:  $\prec_{N_R} \leftarrow$ empty set of priority relationships
6:  $N_R.mapping \leftarrow \texttt{UpdateMapping}(G^m, N_R, e^r)$ for all $e^r \in E^r$
7:  $N_R.cost \leftarrow \texttt{cost}(N_R.mapping)$
8:  $N_R.num\_conf \leftarrow \texttt{count\_conf}(N_R.mapping)$
9:  $STACK \leftarrow \{N_R\}$
10: **while** $STACK \neq \emptyset$ **do**
11:     $N_T \leftarrow STACK.top()$
12:     Remove $N_T$ from $STACK$
13:     **if** $N_T.num\_conf = 0$ **then**
14:         **return** $N_T.mapping$ as solution
15:     $Conf \leftarrow$ chosen conflict in $N_T.mapping$
16:     $C = \{\}$
17:     **for** $(e^r_i, e^r_j)$ involved in $Conf$ **do**
18:         $N^1_C, N^2_C \leftarrow$ copy of $N_T$
19:         $\prec_{N^1_C} \leftarrow \prec_{N_T} \cup \{e^r_j \prec e^r_i\}$
20:         $\prec_{N^2_C} \leftarrow \prec_{N_T} \cup \{e^r_i \prec e^r_j\}$
21:         **for** $N_C \in \{N^1_C, N^2_C\}$ **do**
22:             $success \leftarrow \texttt{UpdateMapping}(G^m, N_C, e^r_{low})$
23:             **if** $success$ **then**
24:                 $N_C.cost \leftarrow \texttt{cost}(N_C.mapping)$
25:                 $N_C.num\_conf \leftarrow \texttt{count\_conf}(N_C.mapping)$
26:                 Insert $N_C$ into $C$
27:     Sort child high-level nodes in $C$, and insert all $N_C \in C$ into $STACK$
28: **return** "No Solution"

29: **Function** $UpdateMapping(G^m, N, e^r_i)$:
30: $SORTED \leftarrow$ topological sorting of the VNR edges $\{e^r_j : e^r_i \prec_N e^r_j\} \cup \{e^r_i\}$ based on $\prec_N$
31: **for** $e^r_j \in SORTED$ **do**
32:     **if** $e^r_j = e^r_i$ or $e^r_j$ conflicts with a higher-priority VNR edge $e^r_k$ **then**
33:         Update $N.mapping$ by invoking a low-level path search for $e^r_j$ to find a $G^m$ path for it that avoids conflicts with all higher-priority VNR edges
34:         **if** unsuccessful in finding a low-level path **then**
35:             **return** $false$
36: **return** $true$
```

In the high-level search, every high-level node N contains a set $\prec_N$ of priority relationships on the G^m paths found for the VNR edges, a VNE mapping $N.mapping$, the cost of the VNE mapping $N.cost$, and the number of conflicts in the VNE mapping $N.num\_conf$. On Lines 4-8, VNE-PBS initializes the root high-level node N_R. On Line 5, it starts with an empty set of priority relationships. On Line 6, it uses the function `UpdateMapping` to find a shortest path for each VNR edge e^r. For N_R, the paths found for all e^r via `UpdateMapping` may conflict with each other since there is no priority relationship on them yet. On Line 7, VNE-PBS calculates the cost of N_R*.mapping*. On Line 8, it counts the number of conflicts in N_R*.mapping*. N_R is then inserted into a first-in-last-out stack of high-level nodes *STACK*.

VNE-PBS expands high-level nodes until either a feasible VNE mapping is found or *STACK* is empty. On Lines 11-12, it retrieves and removes the top high-level node N_T from *STACK*. On Lines 13-14, it returns the feasible VNE mapping N_T*.mapping* if it has no conflicts. Otherwise, it starts to resolve the conflicts in N_T*.mapping*. On Line 15, it chooses the first identified conflict in N_T*.mapping* to resolve. On Line 16, VNE-PBS initializes an empty set C of child high-level nodes. On Lines 17-26, it resolves the chosen conflict, as described earlier. For each child high-level node, it creates a copy of the high-level node N_T, adds a new priority relationship to $\prec_{N_C}$, and updates the G^m paths accordingly. On Line 22, e^r_{low} refers to the VNR edge that is assigned the lower priority between e^r_i and e^r_j. If the update is successful, on Lines 24-25, VNE-PBS computes the cost and the number of conflicts for the child high-level node N_C. On Line 26, it then inserts N_C into the set C of high-level child nodes. After generating all child high-level nodes for the chosen conflict, on Line 27, VNE-PBS sorts the child high-level nodes in non-increasing order of their costs. If two child high-level nodes have the same cost, it uses the number of conflicts for tie-breaking and prefers child high-level nodes with fewer conflicts. It then inserts the sorted child high-level nodes $N_C \in C$ into *STACK*.

Low-Level Search

The function `UpdateMapping` takes three inputs: the augmented SN graph G^m, the current high-level node N, and the VNR edge e^r_i whose path needs to be updated. This function finds shortest G^m paths for the VNR edge e^r_i and all paths with a lower priority than e^r_i that avoid conflicts with all higher-priority paths. On Line 30, it uses a topological sort to obtain a set of VNR edges that are sorted in the order of their priorities. This set contains e^r_i and all other paths with lower priority. On Line 32, it checks if G^m paths require an update, either for being associated with e^r_i or by virtue of conflicting with higher-priority paths.

On Line 33, the function calls the low-level search of PBS. The low-level search is an A^* search that finds a shortest path from v^f_1 to v^f_2 on G^m for implementing the VNR edge $e^r_j = (v^r_1, v^r_2)$ that avoids conflicts with the G^m paths of all VNR edges that have a higher priority than e^r_j.

When generating the neighbors of v^f_1, it returns those SN vertices v^s that satisfy the CPU and geolocation constraints. If v^f_1 has been mapped to an SN vertex v^s in the G^m paths with higher priority, v^f_1 is restricted to be mapped to v^s again. Similarly, it obeys the higher-priority paths when mapping v^f_2 to an SN vertex. When generating the neighbor v^s_2 of an SN vertex v^s_1, it includes SN vertices $\{v^s_2 : \mathrm{BW}((v^s_1, v^s_2)) \geq \mathrm{BW}(e^r)\}$. It also computes the total bandwidth usage on the SN edge (v^s_1, v^s_2) from the higher-priority paths that utilize it. If the sum of the bandwidth usage and the bandwidth requirement of e^r_j is larger than the bandwidth capacity of the SN edge, it does not consider v^s_2 as a neighbor. It also excludes all fictitious vertices other than v^f_2 as neighbors since a path is not allowed to pass through other fictitious vertices. Moreover, it rejects any path from v^f_1 to v^f_2 that maps two different VNR vertices to the same SN vertex since such a path would result in a type-2 vertex conflict.

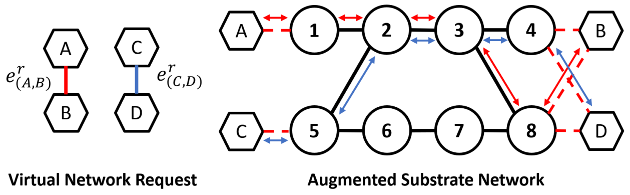

Figure 3: An example of a VNE instance that VNE-PBS fails to solve.

Incompleteness of VNE-PBS

VNE-PBS is incomplete since it does not backtrack on all possible paths for each VNR edge. Figure 3 shows a VNE instance that is optimally solvable with prioritized planning but requires not only using the correct total priority ordering but also breaking ties correctly when planning paths for the VNR edges, which, if done incorrectly, can prevent prioritized planning and VNE-PBS from finding any solution.

In Figure 3, we assume that the CPU requirements of all VNR vertices and the bandwidth requirements of all VNR edges are 1. We also assume that the CPU capacity of all SN vertices and the bandwidth capacity of all SN edges are 1. VNE-PBS starts with the root high-level node that finds the G^m path A-1-2-3-8-B for the VNR edge $e^r_{(A,B)}$ and the G^m path C-5-2-3-4-D for the VNR edge $e^r_{(C,D)}$. This results in a bandwidth capacity conflict at the SN edge 2-3 since its sum of bandwidth usage is 2. To resolve this conflict, VNE-PBS creates two branches that give $e^r_{(A,B)}$ a higher or a lower priority than $e^r_{(C,D)}$, respectively. In the branch where $e^r_{(A,B)}$ has a higher priority than $e^r_{(C,D)}$, there exists no G^m path for $e^r_{(C,D)}$ that does not conflict with the G^m path for $e^r_{(A,B)}$. Similarly, in the branch where $e^r_{(A,B)}$ has a lower priority than $e^r_{(C,D)}$, there exists no G^m path for $e^r_{(A,B)}$ that does not conflict with the G^m path for $e^r_{(C,D)}$. Therefore, both branches are pruned, and VNE-PBS returns no solution. However, VNE-PBS successfully terminates at the root high-level node with an optimal solution if it finds the G^m path A-1-2-3-4-B for $e^r_{(A,B)}$ and the G^m path C-5-6-7-8-D for $e^r_{(C,D)}$.

Experiments

In this section, we present experimental results comparing VNE-PBS against VNE-CBS, G-SP, and RW-MaxMatch-SP. The last two algorithms are popular baseline methods for the core VNE problem. We also experimented

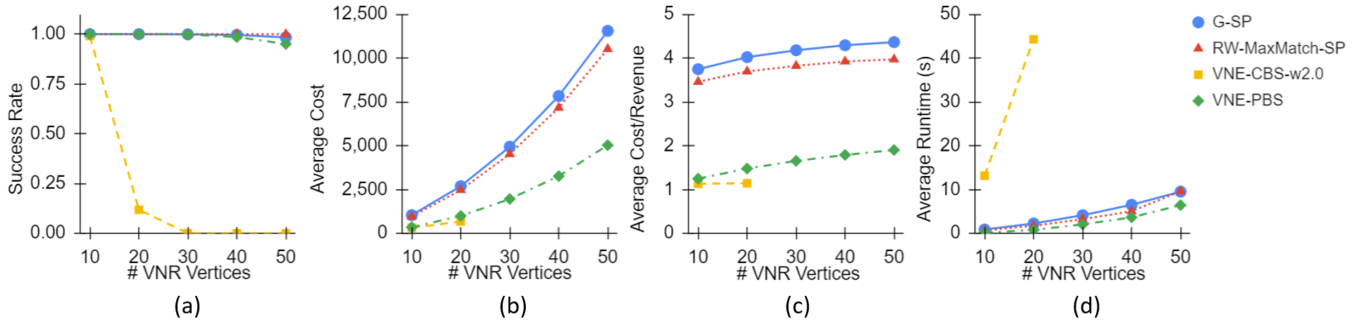

Figure 4: The results of offline experiments averaged over $3,000$ VNE instances for each setting of the number of VNR vertices.

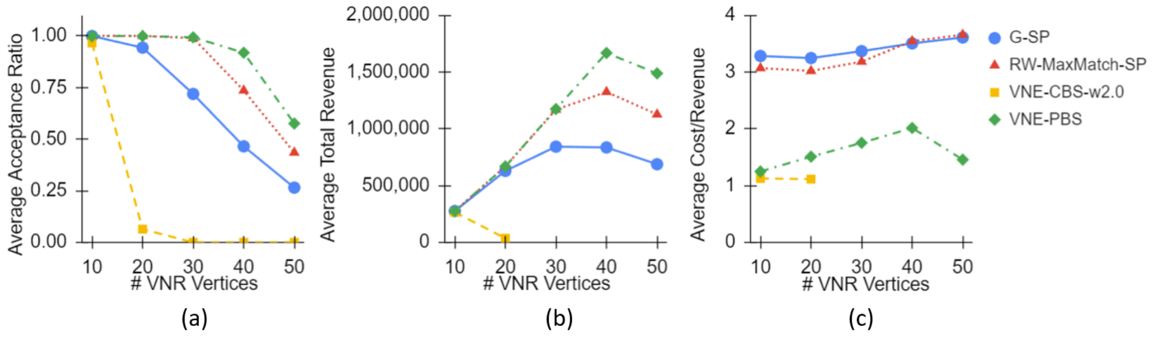

Figure 5: The results of online experiments averaged over 3 runs on each SN for each setting of the number of VNR vertices.

with two other popular baseline methods, D-ViNE and R-ViNE (Chowdhury, Rahman, and Boutaba 2009), but they do not scale well to our large-scale VNE instances. Therefore, we do not include them in the results.

We implemented VNE-PBS and all competing methods in C++ [1]. For VNE-CBS, we used a suboptimality factor $w = 2.0$, the value that gives the shortest runtime as reported in (Zheng et al. 2022). All experiments were run on an AWS machine with 8 CPUs and 16GB RAM. We used a timeout of 60 seconds for embedding a VNR in an SN.

Waxman graphs (Waxman 1988) are commonly used in the VNE literature to simulate communication networks. Our SN topologies were 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. The CPU and bandwidth capacities of the SN vertices and edges are real numbers generated uniformly at random from the interval $[50, 100]$. Our VNR topologies are randomly generated Waxman graphs with $\alpha = 0.3$ and $\beta = 0.2$. We set the number of VNR vertices to be 10, 20, 30, 40, and 50, generating $1,000$ VNRs for each setting. The VNR vertices were located in the same 100×100 grid space as the SN vertices. The maximum allowed Euclidean distance for the geolocation constraints of VNR vertices was set to 15. The CPU requirements of the VNR vertices and the bandwidth requirements of the VNR edges were real numbers drawn uniformly at random from the interval $[0, 20]$ and $[0, 50]$, respectively.

[1]https://github.com/YiZ7699/VNE-PBS

Our SNs and VNRs used in our experiments are significantly larger than those 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 up to with 50 vertices and around 80 edges.

Offline Experiments

In this subsection, we present the results of our offline VNE experiments. For each setting of the number of VNR vertices, we create $3,000$ VNE instances by mapping each of the $1,000$ VNRs to each of the 3 SNs. The reported results are averaged over these $3,000$ VNE instances for each setting of the number of VNR vertices.

Both D-ViNE and R-ViNE show poor performance even on the VNE instances with only 10 VNR vertices, where they managed to solve only 54 and 49 VNE instances, respectively, out of the $3,000$ VNE instances, with an average runtime of around 58.3 seconds on the successful VNE instances. They failed to solve any of the VNE instances with more than 10 VNR vertices. Therefore, we exclude them from the presented experimental results.

Figure 4 shows the following performance metrics: (a) the success rate, representing the percentage of the successfully solved VNE instances; (b) the average cost, representing the cost of the VNE mapping averaged over the successfully solved VNE instances; (c) the average cost/revenue ratio, representing the cost divided by the revenue of the VNE mapping averaged over the successfully solved VNE instances; and (d) the average runtime, representing the runtime averaged over the successfully solved VNE instances.

In general, the success rate of VNE-PBS is very high and much better than that of VNE-CBS-w2.0. Although the other algorithms have high success rates as well, VNE-PBS produces solutions that have significantly lower costs compared to them. A similar trend holds for the average cost/revenue. VNE-PBS also outperforms the other algorithms on the metric of the average runtime, despite some of them being greedy algorithms. Overall, VNE-PBS wins over the other algorithms on all performance metrics.

Online Experiments

In this subsection, we present the results of our online VNE experiments. Here, VNRs arrive 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. This experiment setup is commonly used in the VNE literature to simulate online VNE problems. 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 perform 3 runs corresponding to the 3 SNs, each on the same $1,000$ VNRs for each number of VNR vertices.

Figure 5 presents the comparative performance results of the various algorithms on the metrics of acceptance ratio, total revenue, and cost/revenue, averaged over the 3 runs. The acceptation ratio is the fraction of accepted VNRs. The total revenue is the sum of the revenues of the accepted VNRs. The cost/revenue is the cost divided by the revenue for each accepted VNR.

With respect to the average cost/revenue ratio, VNE-PBS outperforms the other algorithms, indicating that it allocates SN resources to accepted VNRs more efficiently. VNE-CBS-w2.0 outperforms VNE-PBS for small VNRs but is unviable for larger VNRs. With respect to the average acceptance ratio, VNE-PBS outperforms the other algorithms due to the efficient allocation of SN resources to previously accepted VNRs. With respect to the average total revenue, VNE-PBS again outperforms the other algorithms, primarily because of its higher average acceptance ratio.

Conclusions and Future Work

The VNE problem is central to network slicing and network resource management in 5G technologies. It is a constrained optimization problem that models the allocation of resources on heterogeneous physical networks for seamless connectivity. In this paper, we presented VNE-PBS, a successful import from the MAPF literature, to solve the VNE problem efficiently and effectively. VNE-PBS is a prioritized heuristic search algorithm that uses priorities in more sophisticated ways than other algorithms. It not only associates priorities with simultaneously mapping the VNR vertices and edges to SN vertices and paths, respectively, but also explores the space of priority orderings using a systematic depth-first search. Through experiments on large-scale VNE instances, we demonstrated its superiority over competing algorithms on various performance metrics in both offline and online settings.

In future work, we plan to enhance VNE-PBS with intelligent strategies for selecting conflicts for resolution. We also plan to apply VNE-PBS to richer variants of the VNE problem.

Acknowledgments

The research at the University of Southern California was supported by DARPA under grant number HR001120C0157 and by NSF under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, 2112533, and 2121028. The research at Simon Fraser University was supported by NSERC under grant number RGPIN2020-06540 as well as a CFI JELF award. 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

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

Cáp, M.; Vokrínek, J.; and Kleiner, A. 2015. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-Formed Infrastructures. In *International Conference on Automated Planning and Scheduling*, 324–332.

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

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

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

Erdmann, M. A.; and Lozano-Pérez, T. 1987. On Multiple Moving Objects. *Algorithmica*, 2: 477–521.

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

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

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence*, 7643–7650.

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

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

Sturtevant, N. R.; and Buro, M. 2006. Improving Collaborative Pathfinding Using Map Abstraction. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 80–85.

Velagapudi, P.; Sycara, K. P.; and Scerri, P. 2010. Decentralized Prioritized Planning in Large Multirobot Teams. In *International Conference on Intelligent Robots and Systems*, 4603–4609.

Wang, K. C.; and Botea, A. 2011. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research*, 42: 55–90.

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

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

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

Zheng, Y.; Ravi, S.; Kline, E.; Koenig, S.; and Kumar, T. K. S. 2022. Conflict-Based Search for the Virtual Network Embedding Problem. In *International Conference on Automated Planning and Scheduling*, 423–433.

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