BIRD-VNE: Backtrack-Avoidance Virtual Network Embedding in Polynomial Time

Sherif Abdelwahab*, Bechir Hamdaoui*, and Mohsen Guizani†
* Oregon State University, abdelwas,hamdaoui@eecs.oregonstate.edu
† Qatar University, mguizani@ieee.org

***Abstract*—The virtual network embedding (VNE) problem is known to be NP-hard, and as a result, several heuristic approaches have been proposed to solve it. These heuristics find sub-optimal solutions in polynomial time, but have practical limitations, low acceptance rates, and high embedding costs. In this paper, we first propose two heuristics that exploit the constraint propagation properties of the VNE problem to ensure both topological and capacity disjoint consistencies, thereby avoiding backtracking while increasing acceptance rates. Then, combining these two heuristics, we design a polynomial-time VNE algorithm (we term it BIRD-VNE) that, in addition to avoiding backtracking and increasing acceptance rates, incurs a low embedding cost when compared to existing approaches.**

I. INTRODUCTION

Network virtualization emerges as a key technology essential for the success of cloud computing [1]. In general, cloud service providers that adopt the Infrastructure as a Service (IaaS) model rely on virtualization to offer multiple virtual network instances, all embedded into their underlying substrate networks, to multiple different customers. Moreover, with the emergence of the Internet of Things paradigm, cloud-based sensing approaches are foreseen to evolve beyond conventional sensing techniques [2], [3], thereby offering new forms of services. Remote sensing is an example of such new services, which is to be enabled by distributed sensory devices. Network virtualization will play a vital role for promoting these new services [4].

Virtual Network Embedding (VNE) is an important resource management technique in network virtualization. In this context, a typical cloud service provider receives a number of virtual network requests, where each request consists of a set of virtual nodes, each demanding a certain CPU processing capacity, and set of virtual links connecting these nodes, each demanding a certain bandwidth capacity. One key challenge associated with VNE is resource allocation; that is, embedding as many virtual networks as possible while meeting resource demands (both CPU and bandwidth) of all embedded networks.

The effectiveness of VNE can be seen from many perspectives, from meeting contracted SLAs, to maximizing economical profit, to minimizing energy consumption, to ensuring secrecy and quality of service. Another important metric for measuring the effectiveness of a VNE technique is complexity. The VNE problem is known to be NP-hard [5], and the need for polynomial time algorithms is crucial for the success of network virtualization technology.

The recent survey by Fischer et al. [6] presents ongoing and future VNE research efforts and a detailed classification of VNE algorithms. In [7], the VNE problem is formulated as an integer linear program with different objective functions, and solved using backtracking. Lischka et. al. [8] also use backtracking to solve the VNE problem but by formulating it as a graph isomorphism. A major problem with backtracking is that it has an exponential time complexity [9]. In an effort to address complexity, [8] propose to terminate the VNE algorithms when their execution times exceed a defined threshold without bounding the incurred embedding cost.

Heuristic and stochastic algorithms [10] are also proposed, which find sub-optimal solutions but with lesser complexity (polynomial time). For example, in [11], the authors formulate two stage coordinated node and link mapping problems as two mixed integer programming problems, and use rounding relaxation to find near-optimal solutions without rigid upper bounds on the incurred cost. Moreover, in the worst-case scenario, where the virtual network size is comparable to the substrate network size, the time complexity of this algorithm becomes $O(n^{14}\, b^2\, \ln b\, \ln\ln b)$ where n is the number of substrate nodes, and b is the number of input bits to the linear program [11].

In this paper, we propose an *approximation*, *backtrack-avoidance*, and *polynomial-time* VNE algorithm (we name it BIRD-VNE). By studying the constraint propagation properties of the VNE problem, we design a topological consistency algorithm, and show that it achieves a backtrack-free search if substrate paths are all *capacity disjoint*. Then, we relax the capacity disjoint condition, and propose a capacity disjoint paths consistency algorithm that improves the acceptance rate. Finally, we combine these two proposed algorithms to design a *backtrack-avoidance* and $O(n^7)$ VNE algorithm. Our proposed algorithm achieves an embedding cost that, at most, is twice the optimal cost obtained by branch and bound.

The rest of this paper is organized as follows. We formalize and state the VNE problem in Section II. Then, we propose topological consistency (TOPOLOGY-CONSISTENCY) and capacity disjoint paths consistency (CAPACITY-DISJOINT) algorithms in Section III, and show how these algorithms lead to almost backtrack-free search. In Section IV, we propose our VNE algorithm (BIRD-VNE) and derive bounds on its incurred cost. Additionally, we evaluate performance of our BIRD-VNE algorithm empirically in Section V. Finally, we conclude the paper in Section VI.

II. PROBLEM FORMULATION

The substrate network is modeled as an undirected graph $\Phi = (S, L)$ where S is the set of all substrate nodes and L is the set of all substrate links; i.e., each link $l \in L$ corresponds to a connected pair of nodes $s, s' \in S$. We

This work was supported in part by the National Science Foundation, NSF CAREER award CNS-0846044.

assume that each node $s \in S$ offers a processing capacity $C(s)$, and each link $l \in L$ offers a bandwidth capacity $C(l)$. In what follows, let $n = |S|$ and $m = |L|$. Let $\mathbf{R}$ be the set of all possible paths between all substrate node pairs, where a path $P(s_0, s_r)$ between two substrate nodes s_0 and s_r is a sequence $(s_0, s_1), (s_1, s_2), \ldots, (s_{r-1}, s_r)$ of distinct links (or pairs of nodes) in L in which each pair $(s_i, s_{i+1}), i = 1, ..., r-1$, maps to a link in S. Throughout, $P(s_0, s_r)$ (or sometimes P) will also be used to refer to the set $\{(s_0, s_1), (s_1, s_2), \ldots, (s_{r-1}, s_r)\}$, and will be characterized by its length $|P|$ and its bandwidth capacity $C(P) = \min_{l \in P} C(l)$.

Recall that multiple different VNE requests are to be received in real time, and up on receipt of each request, the received virtual network is to be embedded into the substrate network. Each VNE request can be modeled as an undirected graph $\Upsilon = (V, E)$ where V is the set of virtual nodes and E is the set of virtual links (i.e., connected pairs of virtual nodes). We assume that each node $v \in V$ has a requested processing capacity (referred to as node stress) $T(v)$, and each virtual link $e \in E$ has a requested bandwidth capacity (referred to as link stress) $T(e)$.

Suppose that, at a given point in time, a total of $k-1$ VNE requests, $\Upsilon^{(1)}, \Upsilon^{(2)}, \ldots, \Upsilon^{(k-1)}$, have already been received and embedded successfully into the substrate network, and the k^{th} request, $\Upsilon^{(k)}$, has just arrived. The problem of embedding of the k^{th} virtual network $\Upsilon^{(k)} = (V^{(k)}, E^{(k)})$ into the substrate network Φ consists of two mapping steps, node mapping and link mapping, described as follows:

Node mapping, which consists of mapping each virtual node $v \in V^{(k)}$ to a distinct substrate node $s \in S$ such that (i) s is within Δ distance from v and (ii) the sum of requested processing capacities of all virtual nodes mapped to s (including those mapped from previous VNE requests) does not exceed the offered processing capacity of s. Formally, this consists of finding a node mapping function, $\mathcal{M}\left(V^{(k)}\right) : v \in V^{(k)} \mapsto \mathcal{M}(v) \in S$, such that $\mathcal{M}(v_i) = \mathcal{M}(v_j)$ iff $v_i = v_j$, $Dist(\mathcal{M}(v), v) \leq \Delta$ for all $v \in V^{(k)}$, and $\sum_{v \in \cup_{i=1}^{k} V^{(i)}: \mathcal{M}(v) = s} T(v) \leq C(s)$ for all $s \in S$. The parameter Δ is specified by the VNE request and represents a constraint that the embedded network needs to meet. $Dist(u, v)$ represents the Euclidian distance between nodes u and v.

Link mapping, which consists of mapping each virtual link $e \in E^{(k)}$ to a substrate path $P \in \mathbf{R}$ such that (i) the end virtual nodes of e correspond to the end substrate nodes of P and (ii) for every $l \in L$, the sum of requested bandwidth capacities of all virtual links (including those belonging to previous VNE requests) whose mapped paths go through the substrate link l must not exceed the offered bandwidth capacity of l. Formally, this consists of finding a link mapping function, $\mathcal{M}\left(E^{(k)}\right) : e = (v, v') \in E^{(k)} \mapsto \mathcal{M}(e) = P(s, s') \in \mathbf{R}$, such that $\mathcal{M}(v) = s$, $\mathcal{M}(v') = s'$, and $\sum_{e \in \cup_{i=1}^{k} V^{(i)}: l \in \mathcal{M}(e)} T(e) \leq C(l)$ for all $l \in L$.

Definition 2.1: The embedding of $\Upsilon^{(k)}$ is said to be feasible when both the node mapping and link mapping tasks defined above can be performed successfully.

Every time a virtual network request is successfully embedded, the substrate network updates its offered processing and bandwidth resources. Hereafter, we denote the remaining processing capacity of substrate node s, after successfully embedding the k^{th} virtual network request, by

$$R^{(k)}(s) = C(s) - \sum_{v \in \cup_{i=1}^{k} V^{(i)}: \mathcal{M}(v) = s} T(v)$$

Similarly, we denote the remaining bandwidth capacity of substrate link l, after successfully embedding request k, by

$$R^{(k)}(l) = C(l) - \sum_{e \in \cup_{i=1}^{k} V^{(i)}: l \in \mathcal{M}(e)} T(e)$$

and the remaining path capacity of substrate path P by

$$R^{(k)}(P) = \min_{l \in P} R^{(k)}(l)$$

Definition 2.2: For each $v \in V^{(k)}$, we define virtual node v's mapping domain D_v to be the set of all substrate nodes whose Euclidean distances to v are each less than Δ and remaining processing capacities are each greater than $T(v)$; that is,

$$D_v = \{s \in S : Dist(s, v) \leq \Delta, R^{(k)}(s) \geq T(v)\}$$

Definition 2.3: For each $e = (v, v') \in E^{(k)}$, we define virtual link e's mapping domain D_e to be the set of all substrate paths whose end nodes (s, s') are in $D_v \times D_{v'}$ and remaining capacities are each greater than $T(e)$; that is,

$$D_e = \{P \in \mathbf{R} : (s, s') \in D_v \times D_{v'}, R^{(k)}(P) \geq T(e)\}$$

Fig.1 shows an illustrative VNE example, in which the node mapping domains are $D_a = \{A, C\}$, $D_b = \{G, H\}$, and $D_c = \{B, E, F\}$. In the worst case the size of the node mapping domain is $O(n)$. In the figure, link mapping domains are shown in dashed lines. For example,

$$\begin{aligned} D_{(a,c)} = \{&\{(A, B)\}, \{(A, E)\}, \{(A, E), (E, B)\}, \\ &\{(A, B), (B, E)\}, \{(A, C), (C, D), (D, E)\}, \\ &\{(A, C), (C, D), (D, E), (E, B)\}\} \end{aligned}$$

In this same example, the node and link embedding are: $\mathcal{M}(a) = C$, $\mathcal{M}(b) = H$, $\mathcal{M}(c) = B$ for the virtual nodes and $\mathcal{M}((a, b)) = \{(C, B), (B, H)\}$, $\mathcal{M}((a, c)) = \{(C, A), (A, B)\}$, $\mathcal{M}((b, c)) = \{(H, E), (E, B)\}$ for the virtual links.

Our objective in this work is to develop an efficient algorithm that embeds virtual networks into the substrate network; i.e., finds feasible embedding for each VNE request. We say that a feasible embedding is optimal when its cost is minimal, where the embedding cost is defined as the sum of all substrate resources allocated to the virtual network Υ multiplied by a resource unit cost. Formally,

$$Cost(\Upsilon) = \sum_{v \in V} \alpha' T(v) + \sum_{e \in E} \beta' T(e) \times |\mathcal{M}(e)| \quad (1)$$

where α' and β' denote the cost of processing and bandwidth resource units, respectively.

We also define the revenue to be generated from successfully embedding $\Upsilon^{(k)}$ as the sum of all substrate resources

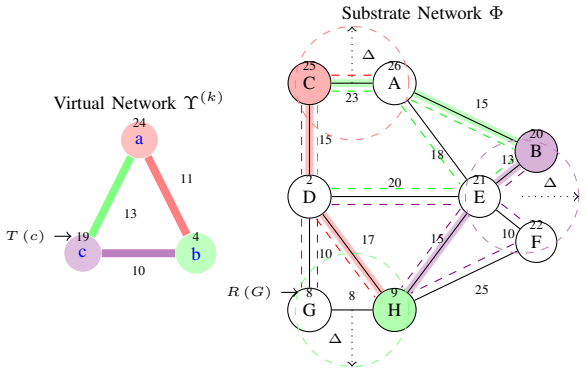

Fig. 1: Virtual Network Embedding: node mapping domains are shown in dashed circles (radius=Δ) and link mapping domains are shown in dashed lines parallel to substrate paths.

requested by the virtual network multiplied by the unit price of each of these resources. Formally,

$$Revenue\left(\Upsilon\right) = \sum_{v \in V} \alpha T\left(v\right) + \sum_{e \in E} \beta T\left(e\right) \qquad (2)$$

where α and β denote the price to be charged for each processing and bandwidth unit, respectively.

III. Enforcing Domain Consistency

Backtracking embedding algorithms involve consecutive node and link mappings while searching for the minimum cost embedding. If any node mapping or link mapping is not successful, these algorithms backtrack to a previous step(s) and attempt to proceed with different mappings.

Backtracking can be avoided (backtrack-free search) if the mapping domains of all virtual nodes and links are consistent, where consistent mapping domains implies that any mapping of a virtual link shall not invalidate any subsequent virtual link mappings. Enforcing domains consistency to ensure backtrack-free search is hard, and general consistency propagation algorithms for this purpose have complexity bounded exponentially in the size of the substrate network (defined by n and m). Instead, we use properties of the embedding problem to narrow the mapping domains by removing mappings which cannot be extended to a complete feasible embedding and show how this leads to an almost backtrack-free search.

A. Topological consistency

1) Alldifferent virtual node mapping constraint: The constraint to map virtual nodes to distinct substrate nodes is known as the alldifferent constraint in the constraint programming context, and we next state a useful corollary following from Régin's theorem [12] on alldifferent constraint.

Corollary 3.1: A virtual node mapping $v \mapsto s$ does not lead to a feasible embedding if it does not belong to a maximum cardinality matching that covers all the virtual nodes in the bipartite graph. In this graph, all virtual and substrate nodes are represented by vertices, and a virtual node v is connected to a substrate node s if $s \in D_v$.

The well-known AllDifferent algorithm [12] can be used to remove mappings from node mapping domains according to Corollary 3.1. For completeness, we next provide a brief description of the AllDifferent algorithm:

1) Construct a bipartite graph B such that
$$B = (V \cup S, \{(v, s) : \mathcal{M}(v) = s\})$$
2) Find a maximum cardinality matching M in B using Hopcroft-Karp algorithm [13].
3) If $|M| < n_v$, then there is no feasible embedding for the given mapping domains.
4) Construct a residual digraph $B' = (V \cup S \cup \{t\}, M \cup E_2 \cup E_3 \cup E_4)$. M is the set of edges in the matching directed from virtual nodes to substrate nodes. E_2 is the set of edges that are not in the matching M and are directed from substrate nodes to virtual nodes. E_3 is the set of all directed edges from substrate nodes in the matching M to a dummy node t. Finally, E_4 is the set of all directed edges from t to substrate nodes that are not in the matching M.
5) Compute the strongly connected components in B'.
6) For any edge connecting two different strongly connected components in B' (i.e. corresponding to a mapping from a virtual node v to a substrate node s) and is not in the matching M, it is not possible to extend a partial embedding that involves mapping v to s. Then s is removed from D_v.

Worst-case complexity of AllDifferent is bounded by $O\left(n_v^{1.5} n\right)$ [12].

2) Relational consistency of node and link mapping domains: In the example of Fig.1, although mapping virtual node c to F is feasible, doing so prevents us from finding a mapping to virtual link (b, c), as there is no substrate path between F and any substrate node in the node mapping domain D_b.

From the definition of mapping domains, we can easily observe that if two virtual nodes v, v' are connected by a virtual link e, then the end points of substrate paths in the virtual link mapping domain D_e is a subset of the cross product of the virtual node mapping domains $D_v \times D_{v'}$. We can now rely on this simple observation and the definition of the virtual link mapping domains to conclude the following:

Lemma 3.2: A virtual node mapping $v \mapsto s$ does not lead to a feasible embedding *if* for any link e ending at v, there is no path $P \in D_e$ ending also at s. Similarly, a virtual link mapping $e = (v, v') \mapsto P(s', s)$ does not lead to a complete feasible embedding *if* $s \notin D_v$ or $s' \notin D_{v'}$.

Proof: Assume $v \mapsto s$ and a subsequent mapping of $e = (v, v')$ such that there is no path $P \in D_e$ ending at s. A mapping of e to any substrate path in D_e results in mapping multiple virtual nodes to the same substrate node. Also, $e = (v, v') \mapsto P(s, s')$ violates the link mapping Definition 2.3 if either $s \notin D_v$ or $s' \notin D_{v'}$. ■

Using Lemma 3.2, we propose two procedures to narrow down the node and link mapping domains: Procedures 1 and 2. The functions $u\left(D_e\right)$ and $v\left(D_e\right)$ return respectively the first and the second end nodes in the link mapping domain D_e. When applied to a path P, $u\left(P\right)$ and $v\left(P\right)$ return the path's first and the second end nodes.

3) Consistency of virtual nodes and substrate nodes connectivity: The relational consistency of node and link mapping domains does not ensure connectivity of the virtual network,

Procedure 1 NODE-CONSISTENCY

Input: $E,\ D_{e\in E},\ D_{v\in V}$
Ensure: Virtual node mapping domains are consistent with virtual link mapping domains in $O(m_v\, n)$
1: **for all** virtual link $e = (v, v') \in E$ **do**
2: $\quad D_v \leftarrow D_v \cap u\,(D_e)$
3: $\quad D_{v'} \leftarrow D_{v'} \cap v\,(D_e)$
4: **end for**
5: **return** Narrowed virtual node mapping domains

Procedure 2 LINK-CONSISTENCY

Input: $E,\ D_{e\in E},\ D_{v\in V}$
Ensure: Virtual link mapping domains are consistent with virtual node mapping domains in $O(m_v\, n^2)$
1: **for all** virtual link $e = (v, v') \in E$ **do**
2: $\quad$ **for all** substrate path $P \in D_e$ **do**
3: $\quad\quad$ **if** $u\,(P) \notin D_v \vee v\,(P) \notin D_{v'}$ **then**
4: $\quad\quad\quad D_e \leftarrow D_e \setminus \{P\}$
5: $\quad\quad$ **end if**
6: $\quad$ **end for**
7: **end for**
8: **return** Narrowed virtual link mapping domains

nor does it imply that the mapping domains can satisfy connectivity requirements, especially when node mapping domains overlap. To illustrate this, consider a new induced network of substrate nodes that represents the connectivity of the virtual link domains. In this induced network, substrate nodes are connected by an edge if there exists a path belonging to any link mapping domain that connects them.

Definition 3.1: Given a virtual network Υ, we define the induced network I of Υ as the undirected graph $I = (S_I \subset S, L_I)$ where $S_I = \cup_{v\in V} D_v$ and $L_I = \{(s, s') \in S_I^2 : \exists\ P(s, s') \in D_e \text{ for } e \in E\}$.

Definition 3.2: For every connected component CC_I of I, the set $N_v(CC_I) = CC_I \cap D_v$ corresponding to the virtual node v is called supernode of v. Let $\zeta(CC_I)$ be the number of distinct supernodes in CC_I. For every $s \in S_I$, we define $\delta(s)$ as the number of supernodes connected to s.

Fig.2 illustrates the induced network of the example given in Fig.1. This induced network is constructed by connecting a pair of substrate nodes in Fig.2 when there is at least one path connecting them in any virtual link domain. In general, if the mapping $v \mapsto s$ is feasible, the function $\delta(s)$ reflects the

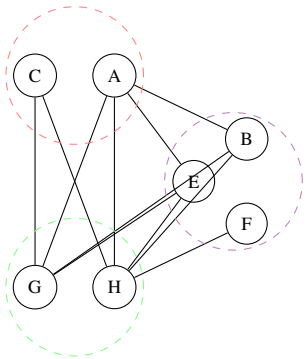

Fig. 2: Induced network I from substrate network Φ in Fig.1. I has one connected components CC_I and three supernodes (dashed circles). $\zeta(CC_I) = 3$, $\delta(F) = 1$ and equals 2 for all other nodes.

degree of the virtual node v, and if a connected component CC_Υ of Υ is mapped to a subset of substrate nodes in Φ, the function $\zeta(CC_I)$ reflects the number of virtual nodes in the connected component CC_Υ (size of CC_Υ).

Lemma 3.3: Let $Deg_\Upsilon\,(v)$ denote the degree of virtual node v. A virtual node mapping $v \mapsto s$ does not lead to a feasible embedding *if* $Deg_\Upsilon\,(v) > \delta(s)$ or the size of the connected component of Υ (CC_Υ) that contains v is greater than the number of supernodes in CC_I that contains s.

Proof: Assume $v \mapsto s$ and $Deg_\Upsilon\,(v) > \delta(s)$, then there exist at least one virtual link e such that there is no substrate path P in D_e with one of its end substrate nodes equals s. Then, $v \mapsto s$ does not lead to a feasible embedding from Lemma 3.2. If $Deg_\Upsilon\,(v) \leq \delta(s)$ but $|CC_\Upsilon| > \zeta(CC_I)$, then there must exist an unmapped virtual node $v' \in CC_\Upsilon$, while all substrate nodes $s \in CC_I$ are already mapped to other virtual nodes in CC_Υ including v. Since v' must be mapped to one substrate node in CC_I to maintain connectivity, then mapping $v \mapsto s$ does not lead to a feasible embedding. ■

The DEGREE-CONSISTENCY procedure (Procedure 3) is a direct application of Lemma 3.3. DEGREE-CONSISTENCY complexity is bounded by computing $\delta(s)$ for all substrate nodes in the virtual node mapping domains, which is $O(n^2)$.

Procedure 3 DEGREE-CONSISTENCY

Input: $E,\ D_{v\in V}$
Ensure: Degree Consistency in $O(n^2)$
1: **for all** virtual nodes $v' \in V$ **do**
2: $\quad$ **for all** substrate nodes $s' \in D_{v'}$ **do**
3: $\quad\quad$ **if** $Deg_\Upsilon\,(v') > \delta(s')$ **then**
4: $\quad\quad\quad D_{v'} \leftarrow D_{v'} \setminus \{s'\}$
5: $\quad\quad$ **end if**
6: $\quad$ **end for**
7: **end for**
8: **for all** connected component $CC_\Upsilon \in \Upsilon$ **do**
9: $\quad$ **for all** connected component $CC_I \in I$ **do**
10: $\quad\quad$ **if** $|CC_\Upsilon| > \zeta(CC_I)$ **then**
11: $\quad\quad\quad D_{v'} \leftarrow D_{v'} \setminus CC_I,\ \forall v' \in CC_\Upsilon$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: **return** Narrowed virtual nodes domains

Running ALLDIFFERENT, NODE-CONSISTENCY, LINK-CONSISTENCY, and DEGREE-CONSISTENCY procedures for one iteration removes some inconsistent mappings from the node and link mapping domains, but may still leave the mapping domains inconsistent. To ensure consistency, these procedures must run repeatedly until no further removal is possible from node and link mapping domains. Algorithm 1 performs this task, with the objective of ensuring topological consistency of the node and link mapping domains, thereby avoiding backtracking due to topological inconsistency.

The complexity of TOPOLOGY-CONSISTENCY is bounded by the number of times we run the procedure LINK-CONSISTENCY in step 4. This implies a complexity of $O(m_v\, n^2)$ in each iteration. In the worst-case scenario, TOPOLOGY-CONSISTENCY removes one substrate node from one node mapping domain and this corresponds to at least one removal of one substrate path from link mapping domains.

Algorithm 1 TOPOLOGY-CONSISTENCY

Input: $E,\ D_{e\in E},\ D_{v\in V}$
Ensure: Topology Consistency in $O(m_v\, n^3)$
1: **repeat**
2: NODE-CONSISTENCY($E,\ D_{e\in E},\ D_{v\in V}$)
3: ALLDIFFERENT($V,\ D_{v\in V}$)
4: LINK-CONSISTENCY($E,\ D_{e\in E},\ D_{v\in V}$)
5: DEGREE-CONSISTENCY($E,\ D_{v\in V}$)
6: **if** $\exists\ D_{v'} = \emptyset,\ \forall v' \in V\ \vee\ D_e = \emptyset,\ \forall e \in E$ **then**
7: **return false**
8: **end if**
9: **until** No node or link domain is changed
10: **return true**

Hence, it requires at most n iterations to remove all substrate nodes from one node mapping domain, thus returning false[1]. Thus, the complexity of TOPOLOGY-CONSISTENCY is $O(m_v\, n^3)$. But since the maximum number of virtual nodes is the number of substrate nodes; i.e., $n_v \leq n$, then the complexity of TOPOLOGY-CONSISTENCY is $O(n^5)$.

B. Capacity disjoint paths consistency

Let us refer again to the example given in Fig.1 and consider the link mapping sequence $(a,b) \mapsto P(C,H) = \{(C,D),(D,H)\}$ and $(a,c) \mapsto P(C,E) = \{(C,D),(D,E)\}$. The remaining bandwidth of the substrate link (C,D), $R\left((C,D)\right) = 15$, is less than the sum of the links' requested bandwidth capacities, which is $T\left((a,b)\right) + T\left((a,c)\right) = 24$. Hence, this mapping sequence is infeasible. Clearly, a search will not be forced to be backtracked if all substrate paths in the link mapping domains are disjoint. However, constructing the link mapping domains from disjoint paths results in a degradation of the VNE acceptance rate (such a rate reflects the number of virtual networks that can be embedded into the substrate network), as well as in an increase in the embedding cost. Our proposed embedding algorithm does not force paths to be disjoint so as to increase acceptance rate and decrease the embedding cost. Instead, our technique relies on the concept of *capacity disjoint* which will be introduced next.

Definition 3.3: For every substrate link l, let $\bar{D}_e(l) = \{P \in D_e : P\ \ni l\}$ and $\bar{E}(l) = \{e \in E : \bar{D}_e(l) \neq \emptyset\}$. We say that the paths in $\mathbf{R}' = \bigcup_{e\in\bar{E}(l)} \bar{D}_e(l)$ are *capacity disjoint iff* the remaining bandwidth capacity of l is greater than the sum of the requested bandwidth capacities of all virtual links in $\bar{E}(l)$. Formally, the paths in $\mathbf{R}'$ are said to be capacity disjoint *iff* $R^{(k)}(l) \geq \sum_{e\in\bar{E}(l)} T\left(e\right)$.

Lemma 3.4: A virtual link mapping $e \mapsto P$ does not lead to a feasible embedding *if* all the substrate paths in every unmapped virtual link's mapping domain are not capacity disjoint with P.

Proof: If a virtual link $e_i \mapsto P_i$ and in a next mapping step of virtual link e_j, all paths in D_{e_j} are not capacity disjoint with P_i, then any mapping $e_j \mapsto P_j \in D_{e_j}$ will result in at least one substrate link with negative remaining bandwidth. ■

Theorem 3.5: The proposed TOPOLOGY-CONSISTENCY algorithm ensures a backtrack-free search if all substrate paths in all link mapping domains are capacity disjoint.

[1]A more efficient implementation checks the condition in step 6 every time any procedure removes a substrate node/link from a mapping domain.

Proof: It follows from Lemmas 3.2, 3.3, and 3.4 and from Corollary 3.1. ■

An algorithm that aims to ensure a backtrack-free search may remove substrate paths that are not capacity disjoint from the virtual links mapping domains. Although such an algorithm will have a complexity advantage because it is backtrack-free, it degrades the acceptance rate and the cost as it will remove substrate paths that can actually lead to feasible or minimum cost embedding. Apparently, capacity disjoint paths condition is required only for substrate paths that are actually in a incurred embedding. In order to overcome the complexity problem while still minimizing the cost and maximizing the acceptance rate, we propose Algorithm 2 (CAPACITY-DISJOINT), which ensures that substrate paths in link mapping domains are capacity disjoint if they are likely to coexist in an incurred embedding.

The key idea of the CAPACITY-DISJOINT algorithm is to determine the worst case scenario in which the intersecting substrate paths in $\mathbf{R}'$ can become simultaneous mappings of virtual links in $\bar{E}(l)$. These paths are found by applying topological consistency procedures, discussed earlier, on the subsets of link and node mapping domains $\bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$ (Steps 1 to 6), where $\bar{V}(l) \subset V$ is the set of end virtual nodes of virtual edges in $\bar{E}(l)$ and $\bar{D}_v(l) \subset D_v$ is the set of substrate node mapping deduced from $\mathbf{R}'$.

The CAPACITY-DISJOINT algorithm checks if all paths that are common to every substrate link l are capacity disjoint. If not, the algorithm removes first the substrate paths $\bar{D}_e \in \bar{E}(l)$ from the domain of the virtual link(s) e that has the largest link mapping domain size $|D_e|$ (Step 7 to 16). This is to minimize the chances of ending up with an empty link mapping domain, thus maximizing the acceptance rate. Although it is clear that CAPACITY-DISJOINT algorithm does not eliminate backtracking entirely, it substantially reduces its likelihood of occurrence. We evaluate the likelihood of backtracking empirically in Section V.

Algorithm 2 CAPACITY-DISJOINT

Input: $E,\ L,\ D_{e\in E},\ D_{v\in V}$
Ensure: Substrate paths are capacity disjoint if they are likely to coexist in an incurred embedding in $O(m\, m_v\, n^3)$.
1: **for all** $l \in L:\ \exists\ P \in D_{e\in E},\ l \in P$ **do**
2: **repeat**
3: NODE-CONSISTENCY($\bar{E}(l),\ \bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$)
4: ALLDIFFERENT($\bar{V}(l),\ \bar{D}_v \in \bar{V}(l)$)
5: LINK-CONSISTENCY($\bar{E}(l),\ \bar{D}_e \in \bar{E}(l),\ \bar{D}_v \in \bar{V}(l)$)
6: **until** No node or link sub-domain is changed
7: $R'\left(l\right) \leftarrow R\left(l\right)$
8: **for all** $e \in \bar{E}(l)$ ordered ascendingly by $|D_e|$ **do**
9: **if** $\bar{D}_e(l) \neq \emptyset$ **then**
10: $R'\left(l\right) \leftarrow R'\left(l\right) - T\left(e\right)$
11: **if** $R'\left(l\right) < 0$ **then**
12: $D_e \leftarrow D_e \setminus \bar{D}_e(l)$
13: **end if**
14: **end if**
15: **end for**
16: **end for**
17: **if** $\exists\ D_e = \emptyset,\ \forall e \in E$ **then**
18: **return false**
19: **end if**
20: **return true**

The CAPACITY-DISJOINT algorithm uses similar steps to determine possible simultaneous intersecting paths (Steps 2 to 6) for each substrate link l that intersects with some paths. Although these steps are performed on a subset of the mapping domains and it is unlikely to encounter the situation that every substrate link is a common link for all paths (as the substrate network will almost look like a path), the complexity of CAPACITY-DISJOINT is bounded by $O(m\, m_v\, n^3)$. This can be expressed as $O(n^7)$ if both the substrate and virtual networks are complete graphs and have the same number of nodes n.

IV. APPROXIMATE COST MINIMIZATION

TOPOLOGY-CONSISTENCY and CAPACITY-DISJOINT algorithms reduce the search space and improve the running time of backtracking search. However, even in the case of backtrack-free search, an optimal optimization algorithm, like branch and bound, may still traverse the whole search space through brute-force [9].

If we assume that VNE does not involve constraint propagation, it can be solved as the minimum weight matching problem in a bipartite graph. A bipartite graph in this case is the set of virtual links connected by edges to the set of substrate paths. We use this observation to propose Algorithm 3, which finds a VNE such that the incurred embedding cost is at most as twice as the optimal cost.

Our proposed BIRD-VNE algorithm starts by enforcing mapping domain consistency using TOPOLOGY-CONSISTENCY and CAPACITY-DISJOINT. It then searches for an embedding by mapping the virtual links with the greatest demands first to the shortest substrate paths in their domains while ensuring feasible embedding according to Definition 2.1. If any mapping step results in infeasible embedding, the algorithm starts the mapping process over from the first virtual link by assigning it to an unattempted mapping in its domain until a feasible embedding is found or all mappings of the first link are tried out. In Section V, we show empirically how likely a backtrack-free search occurs.

The main loop (Step 10 to 25) has m_v iterations. In the worst-case scenario, for every virtual link, it checks feasible mapping of n^2 paths. The complexity of BIRD-VNE algorithm is bounded by the CAPACITY-DISJOINT complexity $O(m\, m_v\, n^3)$ and can be written as $O(n^7)$ as shown earlier.

V. NUMERICAL RESULTS

The effectiveness of BIRD-VNE is assessed in terms of the metrics suggested in [14]: (i) *Acceptance rate,* defined as the ratio of the total accepted virtual networks to the total requested virtual networks; (ii) *Revenue to Cost ratio (R/C),* defined as $R/C = \sum_{\Upsilon} Revenue\,(\Upsilon) / \sum_{\Upsilon} Cost\,(\Upsilon)$; and (iii) *Average node and link utilization,* defined as $\sum_{s \in S} \frac{R(s)-C(s)}{nC(s)}$ and $\sum_{l \in L} \frac{R(l)-C(l)}{mC(l)}$, respectively.

The likelihood of BIRD-VNE to avoid backtracking is shown empirically by the Backtrack-free ratio metric, defined as the total number of times in which BIRD-VNE finds a feasible embedding by attempting the first mapping of the first virtual link to the total accepted requests.

Algorithm 3 BIRD-VNE

Input: $\Upsilon = (V, E)$, $\Phi = (S, L)$
Require: $D_{\forall e \in E}$, $D_{\forall v \in V}$
Ensure: Embedding $\Upsilon \mapsto \Phi$ in $O(m\, m_v\, n^3)$
1: *SolutionExist* $\leftarrow$ TOPOLOGY-CONSISTENCY
2: *SolutionExist* $\leftarrow$ *SolutionExist* **and** CAPACITY-DISJOINT
3: *SolutionExist* $\leftarrow$ *SolutionExist* **and** TOPOLOGY-CONSISTENCY
4: **if not** *SolutionExist* **then**
5: **return** "No feasible embedding"
6: **end if**
7: **repeat**
8: $\mathcal{M}(e) \leftarrow \emptyset,\ \forall e \in E$
9: **for all** $e = (v, v') \in E$ ordered ascendingly by $|D_e|$, and by $T(e)$ **do**
10: **for all** $P \in D_e$ ordered ascendingly by $|P|$ **do**
11: **if** e is the first virtual link in the order of E **then**
12: $D_e \leftarrow D_e \setminus P$
13: **end if**
14: **if** $e \mapsto P$ result in a feasible embedding **then**
15: $\mathcal{M}(e) \leftarrow P$, $\mathcal{M}(v) \leftarrow u(P)$, $\mathcal{M}(v') \leftarrow v(P)$
16: **break**
17: **end if**
18: **end for**
19: **end for**
20: **until** Feasible embedding is found or all first virtual link mapping domain are attempted.
21: **if** No feasible embedding is found **then**
22: **return** "No feasible embedding"
23: **end if**
24: **return** $\mathcal{M}(e),\ \forall e \in E$ and $\mathcal{M}(v),\ \forall v \in V$

We compare the performance of BIRD-VNE with those of the algorithms proposed in [11], referred to as RVINE-SP (without path splitting) and RVINE-MCF (with path splitting), and with that of the basic Greedy algorithm proposed in [5], referred to as BASELINE. We developed our own event-driven simulator in Python, Simpy and NetworkX. Our simulator also integrates the implementation of RVINE-SP and RVINE-MCF, which are publicly available online[2].

The simulator generates Φ and Υ according to Erdös−Rènyi model. Similar to [11], Φ has 0.5 probability of connecting any two substrate nodes, $n = 50$, $C(s) \sim U(0, 50), \forall s \in S$ and $C(l) \sim U(0, 50), \forall l \in L$. Substrate nodes are placed randomly on a (25×25) grid. The mean inter-arrival time of virtual networks ranges from 5 to 25 networks per time unit, and the average service time is of 1000 networks per time unit. Every Υ has 0.5 probability to connect two virtual nodes, $n_v \sim U(1, 10)$, $\Delta = 15$, $T(v) \sim U(0, 20), \forall v \in V$ and $T(e) \sim U(1, 50), \forall e \in E$. The routing $\mathbf{R}$ is computed during initialization phase.

Fig.3 shows that BIRD-VNE has a higher acceptance rate when compared to the other algorithms. The improvement in acceptance rate is a direct result of Theorem 3.5. Fig.4 shows that the backtrack-free ratio achieved by BIRD-VNE is better than 80% for different arrival rates. BIRD-VNE is likely to find a feasible embedding if it passes the consistency enforcement steps 1 to 6. Any optimization algorithm (even brute-force) cannot find a feasible embedding if TOPOLOGY-CONSISTENCY or CAPACITY-DISJOINT do not succeed.

The revenue to cost ratio of BIRD-VNE algorithm is 20%

[2]http://www.mosharaf.com/ViNE-Yard.tar.gz

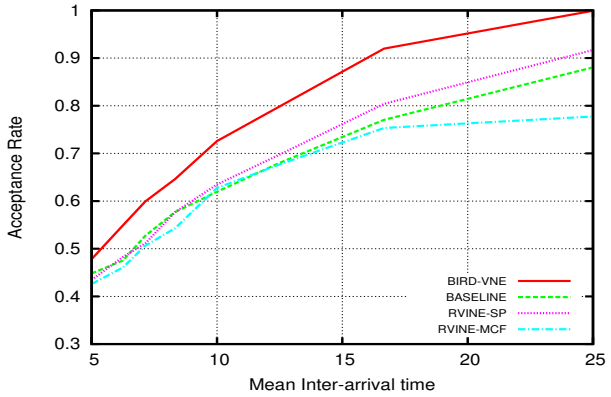

Fig. 3: Acceptance rates under different arrival rates

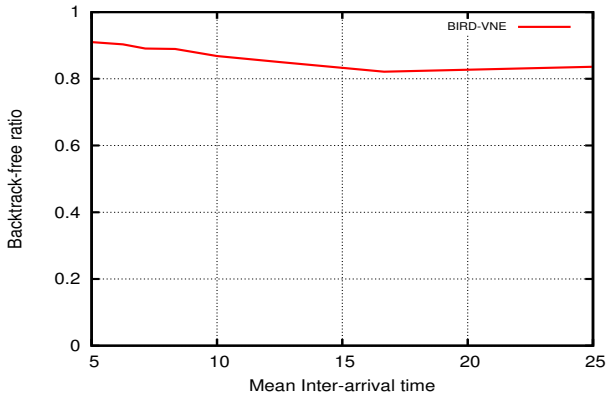

Fig. 4: Backtrack-free rate.

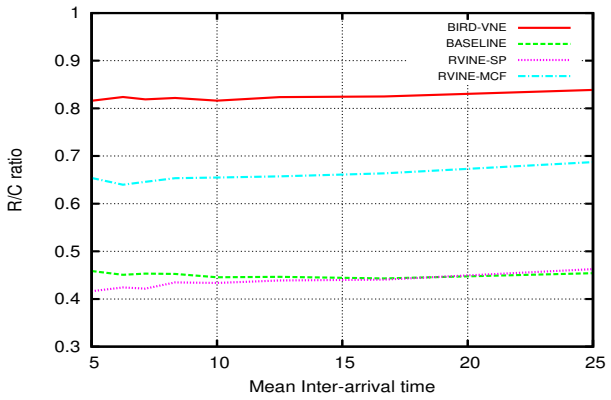

Fig. 5: Revenue to Cost ratio for $\alpha = \beta = \alpha' = \beta' = 1$.

better than RVINE-MCF as shown in Fig.5. This is expected as the algorithm is evaluated empirically as a 2-approximation of the optimal cost, and has better acceptance rate. The average link utilization of BIRD-VNE is comparable to RVINE-MCF with slightly greater node utilization due to better acceptance rate as shown in Fig. 6.

VI. CONCLUSION

We propose a polynomial-time, approximation virtual network embedding algorithm, and show empirically that the algorithm outperforms MIP-based algorithm in terms of revenue to cost ratios as well as acceptance rates. We also show that the likelihood of encountering a backtrack-free search is greater than 80%. This work can be extended to incorporate

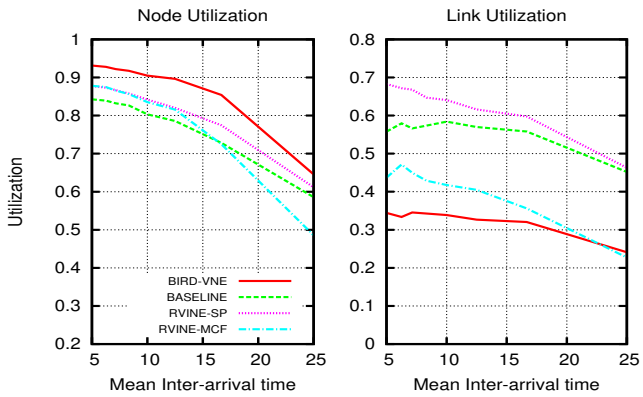

Fig. 6: Average substrate node and link utilization.

migration and path splitting to re-optimize embedding cost and utilization, and improve acceptance rate at higher arrival rates.

REFERENCES

[1] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," *Communications Magazine, IEEE*, vol. 51, no. 7, 2013.

[2] J. Ben-Othman and B. Yahya, "Energy efficient and QoS based routing protocol for wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 70, no. 8, 2010.

[3] L. Mokdad, J. Ben-Othman, B. Yahya, and S. Niagne, "Performance evaluation tools for QoS MAC protocol for wireless sensor networks," *Ad Hoc Networks*, vol. 12, 2014.

[4] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling smart cloud services through remote sensing: an internet of everything enabler," *Internet of Things Journal, IEEE*, vol. 1, no. 3, pp. 276–288, June 2014.

[5] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, 2006.

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

[7] I. Houidi and D. Zeghlache, "Exact adaptive virtual network embedding in cloud environments," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*. IEEE, 2013, pp. 319–323.

[8] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. of ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009.

[9] R. Dechter, *Constraint processing*. Morgan Kaufmann, 2003.

[10] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo, "A unified enhanced particle swarm optimization-based virtual network embedding algorithm," *Int'l J. of Comm. Systems*, vol. 26, no. 8, 2013.

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

[12] J.-C. Régin, "A filtering algorithm for constraints of difference in csps," in *AAAI*, vol. 94, 1994, pp. 362–367.

[13] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[14] A. Fischer, J. F. Botero Vega, M. Duelli, D. Schlosser, X. Hesselbach Serra, H. De Meer *et al.*, "Alevin-a framework to develop, compare, and analyze virtual network embedding algorithms," 2011.