# Cloud Computing – CO466


# A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks

## Submitted to

**Dr. Sourav Kanti Addya**

## Submitted By

Pintu (181CO139)
Vivek Kumar (181CO159)
8th Semester( 4th Year)
Dept. of Computer Science and Engineering NITK, Surathkal

**National Institute of Technology, Surathkal**
**Karnataka**

# Table of Contents

# Abstract

Network virtualization is envisioned to support flexible, cost effective and on-demand deployment of multiple Virtual Networks (VNs) on a shared underlying infrastructure. A key challenge under the virtualization paradigm is how to effectively and efficiently map the divergent VNs onto the shared infrastructure characterized by exhaustible resources. Given that the future services will be characterized by heterogeneity in terms of topology and QoS requirements, existing algorithms can not be flexibly adapted to deal with such requests with differing constraints and mapping objectives. In this regard, this paper proposes a Dynamic Programming Inspired Algorithm (DyPI-Algo), a generic algorithm for mapping virtual networks on a shared infrastructure. Simulation results reveal that the proposed algorithm is able to maximize acceptance ratio and load balancing. Additionally, the algorithm is able to maximise revenue by admitting requests of large size compared to the benchmark algorithms. Moreover, the algorithm is found to be scalable in terms of time complexity when increasing the size of the substrate network and requests. In addition, the paper proposes an Adjacency List based heuristic and a Brute-force algorithm as additional benchmark algorithms. Simulation results show that the proposed algorithms result into up to more than a 10% improvement in terms of acceptance ratio compared to the state of the art algorithms in some scenarios.


# Introduction

The adoption of Network Function Virtualization (NFV) has been widely viewed as a major enabler towards the realization of networks with the ideal flexibility to meet the divergent and stringent future user requirements.The growing interest in the virtualization paradigm is based on two main premises: Firstly, NFV will result in a significant reduction in both capital and operational expenditures incurred by NSPs by enabling multiple logical networks to share the underlying physical infrastructure.Secondly, NFV will lead to an improved Quality of Service (QoS) by granting the end-users or Service Providers (SPs) the flexibility to customize their requests according to the specific requirements of the services to be supported.

The ability to efficiently map multiple VNs onto a shared underlying infrastructure characterized by limited and exhaustible resources. In this regard, the algorithm should optimize the revenue of the NSP by maximizing the number of admitted

requests without violating the QoS of admitted users. And the ability to adapt to changes in request attributes in terms of topology, constraint specification, and embedding objectives, with minimum modification in algorithm's execution strategy.

In this paper, Authors propose a novel dynamic programming inspired algorithm that efficiently maps virtual requests on a shared infrastructure. The proposed algorithm can be adapted to different constraints and objectives with negligible modification in the algorithm execution and construction. Moreover, unfeasible nodes and links do not have any influence on the final solution.

In summary,

1. To reduce the influence of unfeasible nodes and links towards the optimality and time complexity of the final mapping solution, authors propose a candidate extraction algorithm that exploits all node constraints (location, residual resource, etc) and link constraints to extract feasible candidate nodes and links for hosting the request. The rationale behind this approach is that since nodes and links of the VN have associated constraints, these can only be hosted by a subset of underlying nodes and links that satisfy those constraints.
2. To efficiently embed multiple VNs on a shared infrastructure, we propose a Dynamic Programming-Inspired Algorithm (DyPIAlgo) that embeds nodes and links of the request in a coordinated manner. The proposed algorithm exploits a request decomposition technique that transforms a request of any topology into a set of path segments. The proposed algorithm can be adapted to different mapping objectives and constraints with minimal modification.
3. We propose an Adjacency-List based Algorithm (Al-Algo) and a Brute-force algorithm as additional benchmark algorithms. These additional algorithms are found to have a good performance in terms of AR although with additional execution time. The scalability of the proposed algorithms is evaluated in a number of scenarios considering both offline and online cases.
4. Extensive simulations is performed to compare the performance of the proposed algorithms against state of the art heuristics considering a number of performance metrics and scenarios.

# Problem Description

*The problem of embedding virtual networks can be decomposed into two sub-problems:* the virtual node mapping and the virtual link mapping sub-problems . The node mapping stage involves mapping each virtual node of the request onto a suitable substrate node that maximizes the embedding objective while satisfying the virtual node constraints in terms of e.g. location, resource demand, reliability, etc. Similarly, under the link mapping sub-problem, each virtual link is assigned to a suitable substrate link or path that satisfies the virtual link constraints.

The embedding of a VN request onto the underlying infrastructure can be defined as a mapping M from the VNR graph $G^V$ to a subset of the substrate graph $G^S$ such that all the constraints associated with $G^V$ are satisfied. In practice, the goal is not only to satisfy the link and node constraints imposed by the request, but also to optimize a given objective such as revenue of the infrastructure provider , total energy consumption in the substrate network and QoS , among others. Mathematically, the general embedding problem considering the case of VNE can be expressed as follows:

$Optimise$ **Obj** (1)

As shown in Eq. 1, the embedding problem is aimed at maximizing or minimizing a given objective, Obj such as total power consumption, latency and acceptance ratio, among others, while satisfying the request constraints. This is usually done considering a number of constraints some of which are highlighted below:

Node Constraints:

$\forall n^v, m^v \in N^V \quad M^N(n^v) = M^N(m^v)\ if\ f\ n^v = m^v$ (2) $\forall\ n^s \in N^S, n^v \in N^V\ dist(n^s, n^v) \leq dev(n^v)$ (3) $\forall\ n^s \in N^S, n^v \in N^V\ \ T(n^s) = T(n^v)$ (4) $\forall\ t, n^s \in N^S$

$C_t^u(n^s) \leq C(n^s)$ (5)

Link Constraints:

$$\forall\, t, \forall\, q, m \in\ N^S, n \neq m\ Bw_t^u\ (l_{q,m}) \leq Bw(\ l_{q,m}^s) \qquad (6)$$

$$\forall\, m \in\ N^S, l_{i,p}^v \in N^V\ , q \neq m \sum_{q,m \in N^S}\ \ l_{i,p}^{q,m}\ \delta(l_{q,m}^s) \leq \delta_{i,p}(l^v) \qquad (7)$$

**Eq. 2** emphasizes the substrate uniqueness constraint in which no substrate node can host two virtual nodes from the same request where $M^N: N^V \rightarrow N^S$ is a node mapping function. **Eq. 3** is the location constraint while **Eqs. 4 and 5** are the resource constraints. Specifically, from **Eq. 4**, the virtual node can only be embedded on the substrate node of the same resource type. **Eq. 6 and 7** are the bandwidth and delay constraints. From 5, the maximum resource consumption at a given node at any time should not exceed the available resource capacity at that node. From Eq. 6, the consumed bandwidth along a given link at any given time should not exceed the maximum bandwidth on that link. Note however that the nature of the constraints varies usually depending on the attributes considered for the links and nodes and also according to the objective function being optimized. The above problem is computationally hard to solve, hence rendering exact solutions based on solvers such as CPLEX and Gurobi unfeasible for practical delay sensitive applications. In this regard, this paper proposes a heuristic approach with the ability to obtain near optimal solutions with feasible time complexity.

# Dynamic programming inspired algorithm

In this section, we discuss the proposed Dynamic Programming Inspired Algorithm (DyPI-Algo) for performing coordinated node and link mapping. Specifically, we give a detailed discussion of the different steps involved in the execution of the algorithm, including a numerical example on how these are applied. The proposed DyPI-Algo draws two main inspirations from the dynamic programming framework : 1) First, the ability to break up a large, unwieldy problem into a series of smaller tractable problems. In this line, the main idea behind our DyPI-Algo algorithm consists of transforming an otherwise complex request graph into a set of simple edge disjoint path segments whose embedding solutions are computationally tractable. Moreover, performing such a transformation makes the DyPIAlgo and the complementary algorithms suitable for mapping requests of any topology; 2) Second, the ability to traverse a layered network graph in an efficient manner. In this line, our DyPI-Algo implements an embedding technique which involves traversing a multi-layered graph in both forward and backward direction.

Flow chart of entire mechanism:

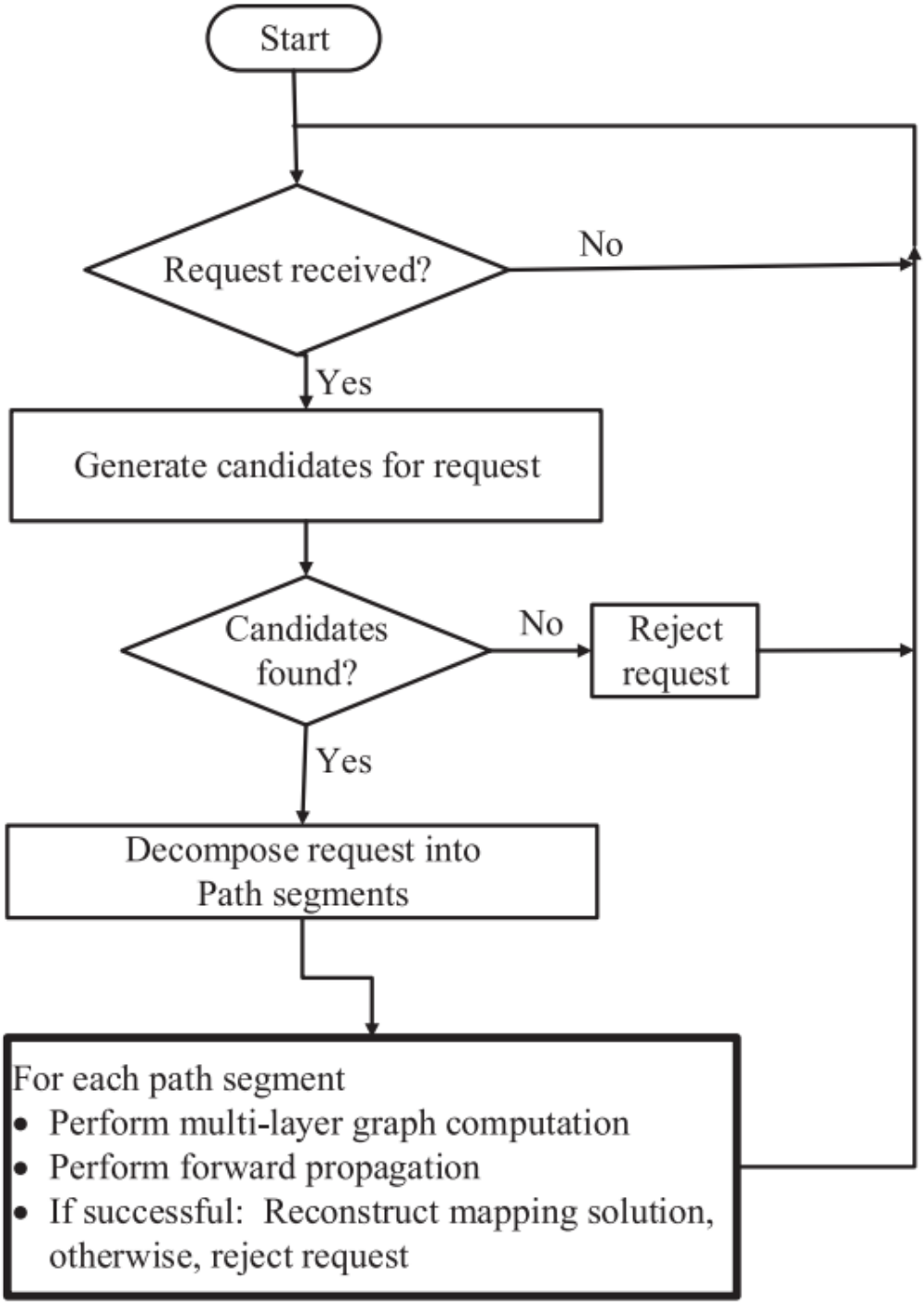



***Fig. A flow-chart illustrating the execution steps of the proposed DyPI-Algo.***

Specifically, the five major steps involved in the algorithm execution are: Candidate extraction step; Request decomposition step; Multi-layer graph computation; Forward propagation step; and Backward/Solution construction step. The candidate extraction step associates each virtual node of the request with a unique set of substrate nodes that satisfy the constraints of that virtual node. These candidate substrate nodes form part of the nodes for the multi-layer graph corresponding to the path segment to which this virtual node is part. The main benefit of this step is twofold: First, the number of nodes constituting the multi-layer graph is significantly reduced. This results in a reduced search space, decreasing the computational time; Second, it guarantees that unfeasible substrate nodes do not influence or leak into the final solution during the solution construction step.

## Candidate Extraction Step

The different requests received by the NSP are usually constrained in terms of acceptable geographical location, type and amount of resources and acceptable probability of failure, among others. Consequently, each virtual node and link of the request can only be hosted by a subset of the underlying nodes and links that can satisfy those constraints. With this motivation, the candidate extraction step aims to associate each virtual node i of the request with a unique set, $CaSn\ i$ , consisting of those substrate nodes that satisfy all the virtual node constraints. For a given substrate node ns ∈ NS, to belong to the candidate set $CaSn\ i$ , such a node must have resources of the type required by the corresponding virtual node and these must be sufficient to satisfy the resource demand. Moreover, such a node must satisfy any other constraints associated with the virtual node in terms of acceptable location radius, holding priority and acceptable failure probability, among others. In this work, we consider virtual nodes to be constrained by resource type, resource amount and location. Therefore, a candidate substrate node for virtual node i should

**Algorithm 1** Candidate Extraction algorithm.

Input: $G^S, G^V$

Output: Set of candidate sets, $cand_s^{req^k}$

**procedure** VIRTUAL NODE CANDIDATE SELECTION(Loop $N^V$)
Initialise: $cand_s^{req^k} = \emptyset$ ▷ Initialise set of candidate sets
    **for** Each virtual node $i \in N^V$ **do**
        $CaS_i^n = \emptyset$ ▷ Initialise candidate set
        **for** For each substrate node $n^S \in N^S$ **do**:
            **if** $dist(n^S, i) \leq dev(i)$ and $C_i^a(n^S) \geq C(i)$ **then**
                Add $n^S$ to $CaS_i^n$ ▷ $n^S$ is a possible candidate
            **end if**
        **end for**
        **if** $CaS_i^n = \emptyset$ **then**
            **Reject** request
            break
        **else**
            Add $CaS_i^n$ to $cand_s^{req^k}$
        **end if**
    **end for**
    **for** Each candidate set $CaS_i^n \in cand_s^{req^k}$ $\forall i \in N^v$ **do**
        **for** candidate $n^S \in CaS_i^n$ **do**
            $validate\_candidate()$
        **end for**
        **if** $CaS_i^n = \emptyset$ **then**
            **Reject** request
            break
        **else**
            Add $CaS_i^n$ to $cand_s^{req^k}$
        **end if**
    **end for**
    $candidate\_uniquify(cand_s^{req^k})$
    Return $cand_s^{req^k}$
**end procedure**

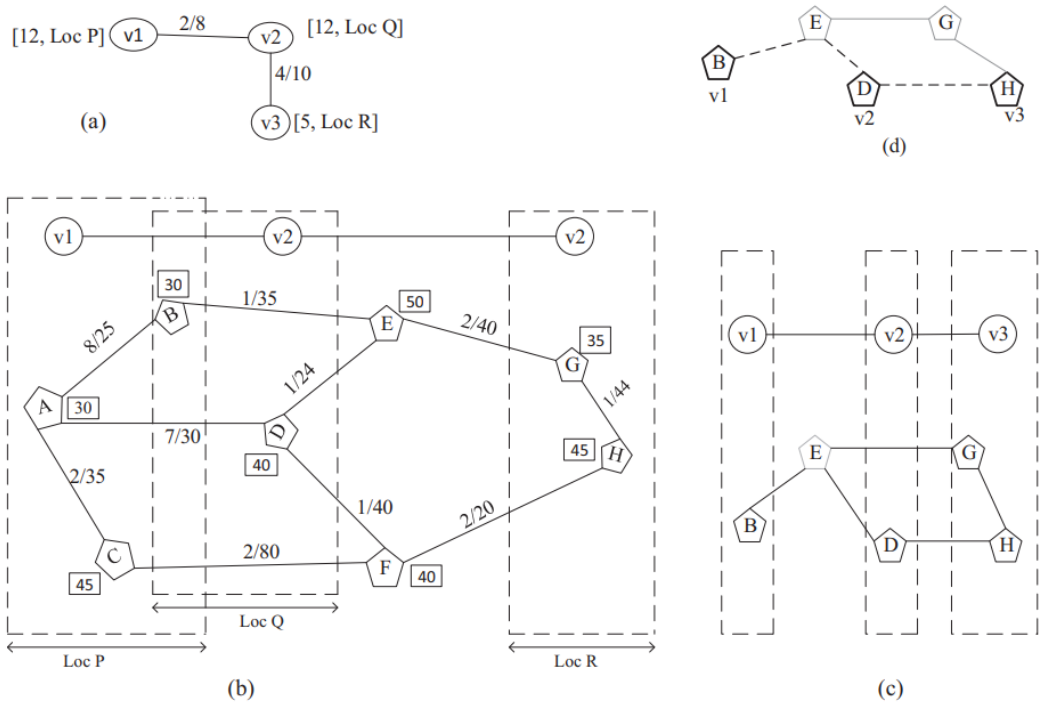



**Fig. 1.** Single substrate embedding illustration:(a) VNR with virtual node and link requirements; (b) The VNR nodes with vertical boxes indicating the possible candidate substrate nodes by virtue of cpu and location constraints; (c) Validated possible candidate substrate nodes and links for the different virtual links and nodes of the VNR (d) Final embedding decision.

**Algorithm 2** Forward propagation step.

Input: Multi-layer graph, $G_{\rho_s}$

Output: Multi-layer graph $G_{\rho_s}$ with global states

1: **for** Each layer $i \in \{1, 2, 3.I\}$ $G_{\rho_s}$ **do**
2: **while** $i < I$ **do**
3: **for** Each node $m \in CaS_{i+1}^m$ at layer $i+1$ **do**
4: **for** Each node $n \in CaS_i^n$ at $i$ with global state S(n) **do**
5: Propagate S(n) to $m \in CaS_{i+1}^m$
6: Compute resulting global state $S(m)$ for node $m$
7: **end for**
8: **end for**
9: **if** S(m) = 0 $\forall$ $m \in CaS_{i+1}^m$ **then**
10: **Reject** request
11: **end if**
12: **end while**
13: **end for**

**Request decomposition**

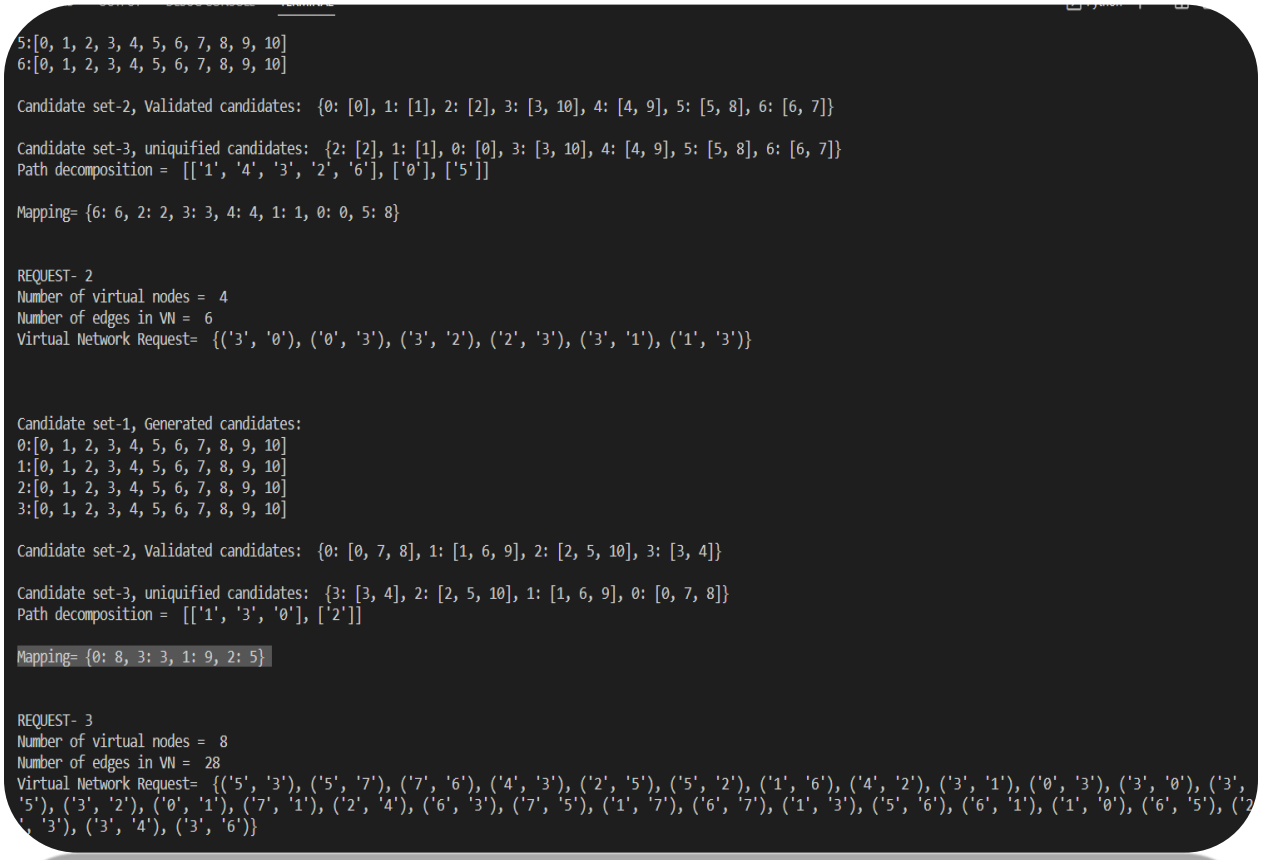
For each request to be embedded, the request transformation into edge-disjoint path segments starts by computing the longest simple path in the request graph and storing this path in the list of possible path segments. Then, the edges constituting this path are removed from the graph. The longest path is again computed considering the remaining edges in the residual graph and removing the used edges from the remaining graph. This process continues until the residual graph is empty. The number of layers in the graph is equal to the number of virtual nodes in the path segment, with the nodes in each layer being the candidate substrate nodes for the virtual node corresponding to that layer. The connections between the adjacent layers of the graph correspond to the weighted substrate paths between the candidate substrate nodes of the adjacent virtual nodes.

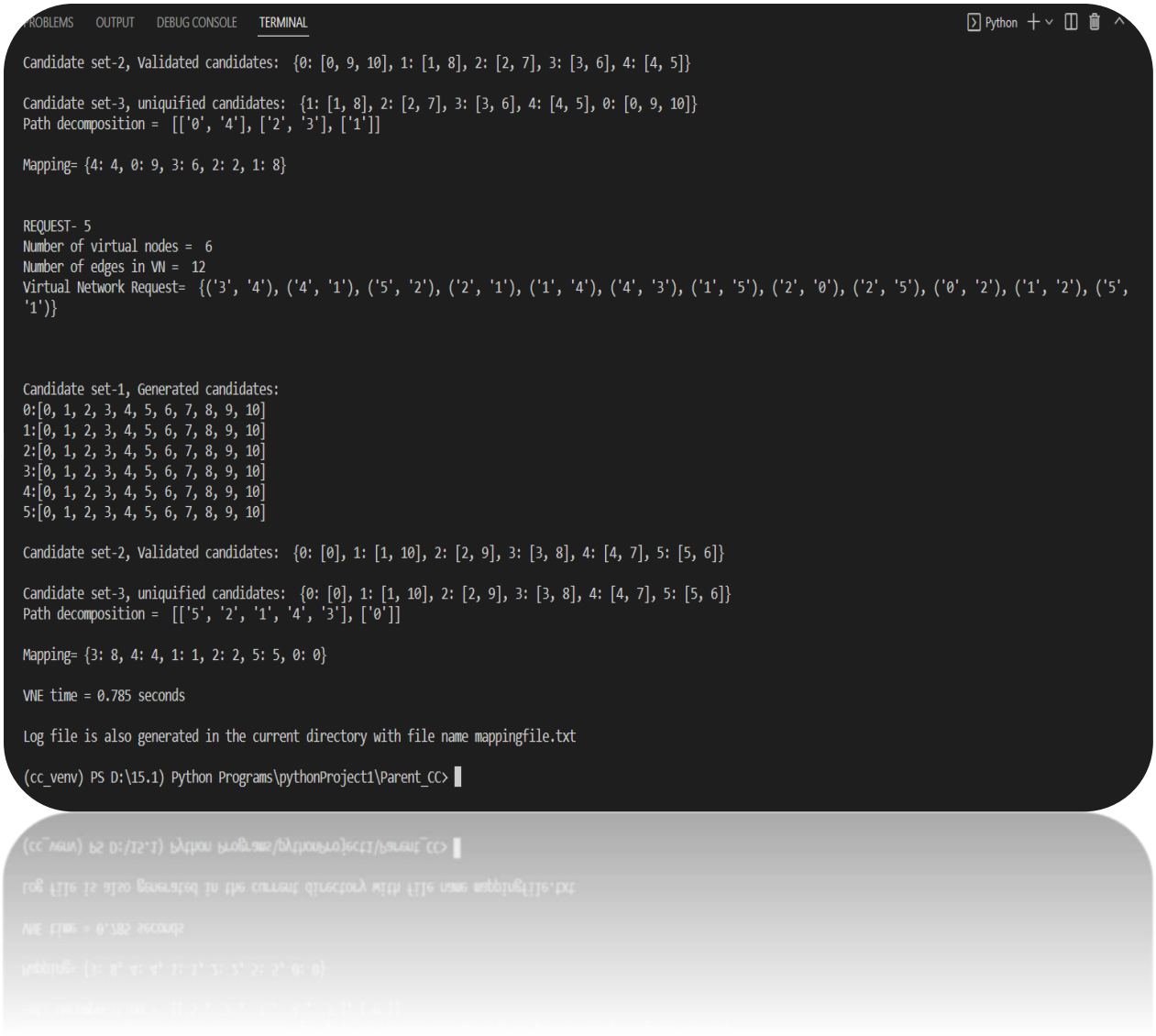
**Solution construction**

Starting with the last layer in the graph, the substrate node with the highest global state is selected to host the last virtual node in the path segment. Consequently, the candidate substrate node from the preceding layer that made the highest contribution to this global score is chosen as the host for the virtual node corresponding to the preceding layer. The substrate link or path connecting these two nodes then becomes the final substrate link to host the virtual link for the virtual nodes mapped onto these nodes. This process of backward propagation continues with the node that made the highest contribution to this layer being selected as the host node for the virtual node of the preceding layer until the host node of the input layer is chosen.

***The*** final embedding solution is computed by traversing the graph backwards from the output layer towards the input layer. Substrate node H is selected to host v3 since it has the highest global state in the output layer. The highest contribution towards H from layer v2 was from substrate node D, hence, D becomes the selected node for the virtual node corresponding to the preceding layer which is v2 in this case.

## Code Output:


5:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
6:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Candidate set-2, Validated candidates: {0: [0], 1: [1], 2: [2], 3: [3, 10], 4: [4, 9], 5: [5, 8], 6: [6, 7]}
Candidate set-3, uniquified candidates: {2: [2], 1: [1], 0: [0], 3: [3, 10], 4: [4, 9], 5: [5, 8], 6: [6, 7]}
Path decomposition = [['1', '4', '3', '2', '6'], ['0'], ['5']]
Mapping= {6: 6, 2: 2, 3: 3, 4: 4, 1: 1, 0: 0, 5: 8}
REQUEST- 2
Number of virtual nodes = 4
Number of edges in VN = 6
Virtual Network Request= {('3', '0'), ('0', '3'), ('3', '2'), ('2', '3'), ('3', '1'), ('1', '3')}
Candidate set-1, Generated candidates:
0:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
1:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Candidate set-2, Validated candidates: {0: [0, 7, 8], 1: [1, 6, 9], 2: [2, 5, 10], 3: [3, 4]}
Candidate set-3, uniquified candidates: {3: [3, 4], 2: [2, 5, 10], 1: [1, 6, 9], 0: [0, 7, 8]}
Path decomposition = [['1', '3', '0'], ['2']]
Mapping= {0: 8, 3: 3, 1: 9, 2: 5}
REQUEST- 3
Number of virtual nodes = 8
Number of edges in VN = 28
Virtual Network Request= {('5', '3'), ('5', '7'), ('7', '6'), ('4', '3'), ('2', '5'), ('5', '2'), ('1', '6'), ('4', '2'), ('3', '1'), ('0', '3'), ('3', '0'), ('3', '5'), ('3', '2'), ('0', '1'), ('7', '1'), ('2', '4'), ('6', '3'), ('7', '5'), ('1', '7'), ('6', '7'), ('1', '3'), ('5', '6'), ('6', '1'), ('1', '0'), ('6', '5'), ('2', '3'), ('3', '4'), ('3', '6')}




# Conclusion

The paper introduced a novel generic algorithm that exploits a multi-layer graph and a request decomposition technique to efficiently embed virtual requests of any topology onto a shared substrate network, while coordinating the mapping of the virtual nodes and links. Specifically, the algorithm decomposes each request graph into a set of edge disjoint paths called path segments. Then, for each path segment,

a weighted multi-layer logical graph is constructed, with the weight of the nodes and links reflecting the suitability of these to host the corresponding virtual nodes and links respectively. The nodes and links constituting this graph are those substrate nodes and paths that are potential candidates for hosting the request. Then, the global state of the graph is obtained by propagating the state of each node along its outing links starting from the input layer towards the output layer in the forward propagation phase of the algorithm. Then, the mapping solution for the virtual links and nodes constituting the path segment is obtained by traversing the graph backwards, and selecting at each layer, those substrate nodes and links that lead to the highest global node state value at the output layer. Additionally, the paper proposed an adjacency list based algorithm and a brute-force algorithm as additional potential mapping algorithms. Simulation results have proved that the DyPI-Algo algorithm not only maximizes the number of accepted requests, but is able to admit requests of large size, hence guaranteeing a high revenue compared to the benchmark algorithms, and the considered state of the art algorithms. This is largely attributed to the fact that the proposed algorithm results in low resource consumption while guaranteeing load balancing. Moreover, the algorithm has been shown to be scalable with the increase in size of both the substrate network and arriving requests. Although the proof of concept regarding the algorithm performance has been made considering the virtual network embedding problem which does not permit embedding multiple virtual nodes of the same request onto the same substrate node, the proposed algorithm is well suited for other mapping problems that arise in the virtualization landscape. Moreover, the algorithm can be used for achieving multiple embedding objectives among the different requests with negligible modification in its implementation and execution.

## References

[1] X. Li, M. Samaka, H.A. Chan, D. Bhamare, L. Gupta, C. Guo1, R. Jain, “network slicing for 5g: Challenges and opportunities,”, IEEE Internet Computing 21 (5) (2017) 20–27, doi:10.1109/MIC.2017.3481355. ISSN: 1089-7801

[2] Zhou, et al., “Network slicing as a service: enabling enterprises’ own softwaredefined cellular networks.”, IEEE Commun. Mag. 54 (2016) 146–153, doi:10.1109/CC.2018.8332001.

[3] M. Leconte, G. Paschos, P. Mertikopoulos, U. Kozat, “A resource allocation framework for network slicing,”, IEEE International Conference on Computer Communications (INFOCOM 2018) (2018), doi:10.1109/INFOCOM.2018.8486303.

[4] Y. Choi, N. Park, “Slice Architecture for 5G Core Network,”, in: Nineth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, 2017, pp. 571–575, doi:10.1109/ICUFN.2017.7993854.

[5] F. Malandrino, et al., “reducing service deployment cost through VNF sharing, in: IEEE/ACM Transactions on Networking, 2019, pp. 1–14. Https://arxiv.org/abs/1910.03611

[6] Description of network slicing concept,[Online] Available: https://www.ngmn.org/fileadmin/user_upload/161010_NGMN _Network_Slicing_framework_v1.0.8.pdf.

[7] M. Raza, M. Fiorani, A. Rostami, P. ÃÜhlen, L. Wosinska, P. Monti, “Dynamic slicing approach for multi-tenant 5g transport networks [invited],”, J. Opt. Commun. Netw. 10 (2018) A77–A90, doi:10.1364/JOCN.10.000A77.

[8] M. Chowdhury, M.R. Rahman, R. Boutaba, “ViNEYard: virtual network embedding algorithms with coordinated node and link mapping,”, IEEE/ACM Trans. Networking 20 (1) (2012) 206–219, doi:10.1109/TNET.2011.2159308.

[9] N. Nia, S. Adabi, M. Nategh, “ A coordinated heuristic approach for virtual network embedding in cloud infrastructure,”, KSII Trans. Internet Inf. Syst. 11(5) 2346–2361.

[10] K. Samdanis, S. Wright, et al., “5G network slicing - part 1: concepts, principles, and architectures,”, IEEE Commun. Mag. 55 (5) (2017) 70–71, doi:10.1109/MCOM.2017.7926919.