Optimization of Service Function Assignment and Shortest Path for Network Function Virtualization

Arvind Kalyan
Westview High School
San Diego CA 92129
arvindkrishnakalyan@gmail.com

***Abstract*—This paper focuses on the concept of Network Function Virtualization (NFV): the implementation of requests consisting of various service functions on servers located in data centers. This paper attempts to minimize both the cost of routing and service function assignment of requests from source to destination node on a network. This problem falls under the class of Integer Linear Programming (ILP), which is NP-Hard and cannot be solved in polynomial time. Towards developing a solution, it is proposed to split the problem into two separate optimization subproblems: shortest path routing and service function assignment. We utilize Dijkstra's Shortest Path algorithm and a Greedy method for service function assignment to propose a new heuristic algorithm that minimizes the total cost of routing and service functions assignment. The experimental results suggest that the proposed algorithm matches the optimal ILP solution within acceptable limits.**

Keywords – Network Function Virtualization; Integer Linear Programming; Dijkstra's Shortest Path Algorithm; Greedy Heuristic.

I. Introduction

In the last few years, the rise of powerful new networking technologies and big data has generated a strong need for equally powerful, versatile network services. Internet traffic is higher than it has ever been, and continues to grow in an exponential fashion. The current technologies available to respond to this increased need are inflexible, featuring rigid physical servers that host several separate packages of an operating system with its respective network functions. These functions, such as Deep Packet Inspection, Firewalls, and Content Delivery Network (CDN) Servers, are chained to exactly one operating system, dramatically reducing the efficiency of their host. Servers are forced to run at efficiencies far below their optimal rate, and as the demand for traffic increases has become more and more dynamic in nature, this current technology is simply inadequate. The concept of Network Function Visualization (NFV) has risen as a solution to these presented challenges. NFV, at its core, involves the deployment of several network functions in the form of software on high volume shared servers in data centers. Network controllers dictate the route of this flow from the source node, through the data centers, and to the destination. This concept allows for the programmability of network control, thus allowing the network to adapt flexibly to the required functions. Also, NFV greatly increases the efficiency of each of these servers, and while there remains room for improvement, the decoupling of function and operating system offers versatility with the potential to break through the current limitations. The process of implementing virtual network functions onto a single server and allocating the necessary resources to carry it out is called service function chaining (SFC). SFC creates a chain of different services to be carried out in an appropriate order, taking available resources, size, and other factors into consideration. This process may be automated to provide the fastest and most efficient execution of a set of assigned services, and this has been the focus of much of the previous work regarding the topic.

Section II delves into the contributions of this paper with respect to existing solutions. Section III establishes the mathematical model of the problem, including the constraints and objective function that are to be minimized. Section IV introduces and explains the algorithm we have devised using pseudo-code and a run-time analysis in comparison with existing solutions. The experimental results are graphed and explained in Section V, and Section VI contains the conclusion and final remarks.

II. Related Work and Paper Contribution

Previous authors have utilized an ILP based solution in order to place virtual network functions into appropriate data centers. These authors have also provided heuristics and several approximation algorithms to solve this problem. However, unlike this proposed model, few of them have taken into consideration the routing cost from source to its destination while assigning and placing network functions.

Blenk et al. [1] delved into the placement of hypervisors, which serve as a layer between a Software Defined Network (SDN) controller and networks, consisting of various necessary functions. The authors stated the importance of a "good placement" of these hypervisors. Amaya et al. [2] explored the specifics of service chaining, creating a model that allows various orderings to be ranked and evaluated. They also derive a route from their ordering and apply various constraints to the data centers. Ghasem et al. [3] explored the details of service chaining and the concept of network function virtualization, while Addis et al. [4] focused on the benefits of optimizing the route taken by function virtualization requests. Crichigno et al. [5] considered a routing and placement scheme like the one proposed in this work, however they propose a different heuristic to solve the problem. Subsequent analysis in this paper suggests that the proposed algorithm has superior run time complexity as compared to Crichingo et al. [5]. Cohen,

 ISBN: 978-1-61208-747-4

et al. [6] proposed a near optimal placement of virtual network functions using approximation algorithms guaranteeing a placement with theoretically proven performance. Luizelli et al [7] formulated a network function placement and chaining problem and proposed an ILP model to solve it. Moens et al. [8] proposed a formal model for virtual network function placement (VNF-P) with focus on a hybrid scenario with part of services provided by dedicated physical hardware and the rest provided using virtualized service instances. Gupta et al. [9] provided a mathematical model for the placement of VNFs which ensures the service chaining as required by the traffic flows.

This paper is unique in attempting to explore a solution toward minimizing cost by splitting the problem into two subproblems: optimization of both network service function assignment among data centers and the routing cost. Since the problem is ILP, which is NP-Hard and cannot be solved in polynomial time, we search for a solution of lower complexity. Utilizing a Greedy method and Dijkstra's shortest path algorithm [10][11], we create a heuristic algorithm in order to minimize both costs. The algorithm provides complete source to destination route for all requests along with the assignment of the network functions requested.

III. Mathematical Formulation of the Problem

A. Model

We represent the network using a graph $G = (V, E)$, where V represents the set of nodes on the graph and E represents the links that connect each node. In this set V of nodes, we define derived subset $D \subseteq V$ as the set of data centers where each network function will be virtualized. Each network function is denoted by $f \in F$. A request $r \in R$ from source node src_r to destination node dst_r contains a group of n functions $f_{r,1}, f_{r,2}, \ldots, f_{r,n} \in F_r$ to be implemented by the network. For a pair of nodes $(i, j) \in E$, $c_{i,j}$ is the predetermined cost of traversing that link, from node i to node j. $x^r_{i,j}$ is a binary variable (either 0 or 1) that denotes whether that link is traversed or not by request r. For each data center $d \in D$, W_d is the maximum capacity available on data center d to virtualize the network function. There are many types of resources that may be denoted by this variable, including storage, memory, and CPU cores. We denote the resource required to virtualize network function $f \in F$ from data center $d \in D$ as $w_{d,f}$, and the cost of virtualizing the network function on the data center is denoted by $c_{d,f}$. We define $y^r_{d,f}$ as another binary variable, specifying whether or not a function in f is virtualized or not on a specific data center d on a certain request r.

B. Objective

This problem involves routing and assignment, so it requires two functions to optimize the problem.

First, we aim to minimize the total Service Function Assignment cost across the given request. That is, the sum, across all services $f \in F$ and data centers $d \in D$, of the cost of implementing a given function multiplied by whether or not the service is implemented ($c^r_{d,f} \cdot y^r_{d,f}$).

$$\min \sum_{r \in R} \sum_{f \in F} \sum_{d \in D} c^r_{d,f} \cdot y^r_{d,f} \tag{1}$$

Next, we aim to minimize the routing cost for all requests $r \in R$, given as the sum across all pairs $(i, j) \in E$ of the product of associated cost and whether that link is traversed or not ($c_{i,j} \cdot x^r_{i,j}$).

$$\min \sum_{r \in R} \sum_{(i,j) \in E} c_{i,j} \cdot x^r_{i,j} \tag{2}$$

The overall objective function is to minimize both the service function assignment cost, as well as total routing cost.

$$\min \sum_{r \in R} \sum_{(i,j) \in E} c_{i,j} \cdot x^r_{i,j} + \min \sum_{r \in R} \sum_{f \in F} \sum_{d \in D} c^r_{d,f} \cdot y^r_{d,f} \tag{3}$$

C. Constraints

We constrain decision variable $x^r_{i,j}$ to equal 1 when the link is traversed from node i to node j in a route r and 0 when not traversed:

$$x^r_{i,j} = \begin{cases} 1 \text{ if traversed} \\ 0 \text{ else} \end{cases} \tag{4}$$

Similarly, decision variable $y^r_{d,f}$ is set to 1 when a service function f is assigned to the data center d in a route r and 0 when it is not:

$$y^r_{d,f} = \begin{cases} 1 \text{ if assigned} \\ 0 \text{ else} \end{cases} \tag{5}$$

The third constraint governs the virtualization of each network function f for a route r, allowing each to be implemented only once on a data center.

$$\sum_{d \in D} y^r_{d,f} \leq 1, \forall r \in R, f \in F \tag{6}$$

Additionally, we must ensure the balance of inflow and outflow for each node. Thus, we create a flow constraint for each node traversed in a particular route. The source node must have an outflow of 1, while the destination has an outflow of -1. Every node in between must have a net flow of 0, where the inflow equals outflow.

$$\sum_{(i,j) \in E} x^r_{i,j} - \sum_{(i,j) \in E} x^r_{j,i} = \begin{cases} 1, \ i = src_r \\ -1, \ i = dst_r \\ 0, \ \text{otherwise} \end{cases} \tag{7}$$

Each data center has a maximum ability to implement functions, and we attempt to prevent possible overuse of a data center d by adding a capacity constraint to each d. This constraint prevents each data center from virtualizing functions whose total resource requirement exceeds the given capacity of the center.

$$\sum_{f \in F} w_{d,f} \cdot y^r_{d,f} \leq W_d, \forall r \in R, d \in D \tag{8}$$

 ISBN: 978-1-61208-747-4

We must ensure that there is both an inflow and outflow out of an assigned data center. Therefore, we require the outflow at each data center to be greater or equal to the binary variable denoting its activation. A data center d that virtualizes a function f for route r, for example, will have a $y^r_{d,f}$ value of 1; this constraint requires its outflow to also be at least 1. A data center that is unused, meanwhile, may carry any outflow.

$$\sum_{j \in V} x^r_{i,j} \geq y^r_{d,f}, \forall i \in V, d \in D, r \in R \tag{9}$$

D. Proposed Solution

The above problem belongs to the class of ILP problems, which are NP-Hard problems that cannot be solved in polynomial time. In order to develop a heuristic solution to the problem, we propose splitting the overall problem into two separate optimization subproblems: a) find the shortest route from a source to a destination node, and b) assignment of the network functions through appropriate data center from source to destination node. This novel approach leads us to propose a heuristic that will utilize Dijkstra's shortest path algorithm and a Greedy network function assignment algorithm to solve the problem.

First, we have an assignment problem, with an objective function given by (1). We also utilize (5) governing data center use, (6) limiting service functions to be implemented just once, and (8) that applies the capacity constraint.

We can then model the remainder of the problem as a pure shortest path problem, using Dijkstra's algorithm to develop a solution. For this, we have an objective function given by (2), and constraints (4) and (7).

Finally, we can tie the two problems together using (9), giving us a solution to the overall problem.

IV. Algorithm for Modified Routing and Greedy Assignment

Based on the above problem split, we propose an algorithm using Dijkstra's shortest path and a Greedy heuristic algorithm to assign network functions to solve the problem.

A. Proposed Modified Dijkstra's Algorithm with Greedy Service Function Assignment

We propose a heuristic algorithm that may be utilized on networks. The algorithm requires an input of a graph G with vertices V and edges E. In addition, we specify the costs of implementing a service on each data center ($c^r_{d,f}$) and the cost of routing services from all nodes i to j ($c_{i,j}$). Finally, we set the capacity of each data center (W_d) and the resources taken by each network function on a data center ($w_{d,f}$). The algorithm returns values for $x^r_{i,j}$ which dictate the route taken by the network functions, as well as $y^r_{d,f}$, which denotes the data centers that are assigned to implement each network function. The algorithm also assumes the use of an adjacency matrix to implement the network graph and allow us to easily find the neighbors of a node in the graph through lookup.

We begin by going through each of the requests in a set R. Using the source and the destination nodes of this request, we use Dijkstra's algorithm to discover the shortest path for the request, returning the list of nodes that make up the path as SP_r. From here, we create two separate lists: the first, PDC_r, will store data centers found on the shortest path given by Dijkstra's, while the second list, NDC_r, will store data centers that are neighbors along the shortest path. The shortest path is iterated through and data centers found on the path are added to the first list. Using an adjacency matrix, we define a function $neighbors(n)$ to return all nodes that are linked to node n. Any data centers found among these neighbors are added to NDC_r.

Next, we provide network function assignment for the functions requested by r. For each, we use a Greedy heuristic algorithm in order to select a data center from all those detected by the previous scan, using the list comprised of the union of PDC_r and NDC_r. This is done by comparing the different costs of $c^r_{d,f}$ for each data center and ensuring that the necessary resources are available for implementation. Once the lowest cost data center d has been located, we set the corresponding value of $y^r_{d,f}$ to equal 1, signifying the use of that data center for the function. We then update the available resources of the data center by reducing its capacity W_d by $w_{d,f}$. If the data center is on the path, the shortest path SP_r does not need to be updated. However, if the data center is found from the neighbors list, we insert a one-link detour to that node into SP_r. This process is repeated for every function in the request. This is the key modification to the shortest path algorithm, hence we call it the modified Dijkstra's algorithm.

By the process described above, we expect a majority of the network functions requested to have been assigned and virtualized at an appropriate data center. However, there is a possibility that the request still has unassigned functions, in which case we propose the use of a breadth-first search (BFS) to locate a capable data center. Once a data center is located by BFS, if it contains the necessary resources, we assign to it the remaining functions and update its capacity, $y^r_{d,f}$, and SP_r accordingly. This is the worst-case scenario of the algorithm.

Once all functions have been assigned, we use SP_r to update the values of $x^r_{i,j}$ for the route, setting the variable to 1 when the link is traversed. After this process has been repeated for every request, the algorithm return $x^r_{i,j}$ and $y^r_{d,k}$, for all $r \in R$.

The proposed heuristic algorithm utilizes Dijkstra's shortest path algorithm and a Greedy method in order to calculate an optimal route for a set of requests R. The first component of the algorithm uses the shortest path algorithm to find a path SP_r from the source node src_r to the destination node dst_r. The second uses a Greedy method to find an optimal data center $d \in D$ along the previously discovered path for each service function $f \in F_r$. The Greedy method runs with a linear complexity, while Dijkstra's algorithm implemented with a binary heap will run in logarithmic time. Additionally, the neighbor search is implemented as a simple table lookup due to the adjacency matrix implementation of the network

 ISBN: 978-1-61208-747-4

Algorithm 1 Modified Dijkstra's Algorithm with Greedy Service Function Assignment

Input: $G(V,E); c_{i,j} \forall (i,j) \in E$; $W_d \forall d \in D$; $w_{d,f}, c_{d,f} \forall d \in D$
Output: $y^r_{d,f}, x^r_{i,j} \forall r, d, f$
for all $r \in R$ **do**
  src_r = source node of request
  dst_r = destination node of request
  $SP_r = Dijkstra(src_r, dst_r)$
  initialize PDC_r (set of data centers found on shortest path SP_r) as ϕ
  initialize NDC_r (set of data centers found among path neighbors) as ϕ
  for node $n \in SP_r$ **do**
    if $n \in D$ **then**
      $PDC_r = PDC_r \cup n$
    end if
    for node $m \in neighbors(n)$ **do**
      if $m \in D$ **then**
        $NDC_r = NDC_r \cup m$
      end if
    end for
  end for
  for all $f \in F_r$ **do**
    Select $d \in PDC_r \cup NDC_r$, the data center that implements function f at lowest cost, contains necessary resources (Greedy Assignment)
    set $y^r_{d,f} = 1$
    update resources for d appropriately
    if $d \in NDC_r$ **then**
      update SP_r, insert d
    end if
  end for
  if any $f \in F_r$ remains unimplemented **then**
    run breadth-first search (BFS) starting from the most connected node $\in SP_r$
    if data center d is found by BFS **then**
      implement $f \in F_r$ (provided d has necessary resources available)
      update $y^r_{d,f}$
      insert path to d into SP_r
    end if
  end if
  update all $x^r_{i,j}$ using SP_r
end for
return $x^r_{i,j}, y^r_{d,f}$ $\forall r, d, f$

and is $O(|V|)$. Therefore, we may conclude that the total algorithm is dominated by the first stage Dijkstra's algorithm run-time complexity. The worst case running time of our implementation of Dijkstra's algorithm with a binary heap is given by $O(|E|log|V|)$, and we call the Dijkstra's once for each request R in a scenario. Therefore, our total run time for the proposed heuristic algorithm is $O(|R||E|log|V|)$. Notably, as a comparison, the algorithm of Crichigno et al. [5] utilizes Dijkstra's algorithm twice, resulting in a worst-case run-time of $O(|R||D||E|log|V|)$, where D is the total number of data-centers contained in the topology. Thus the proposed heuristic algorithm is able to run at a faster run time by a factor of $|D|$ as compared to the algorithm proposed by Crichigno et al. [5].

V. Experimental Results

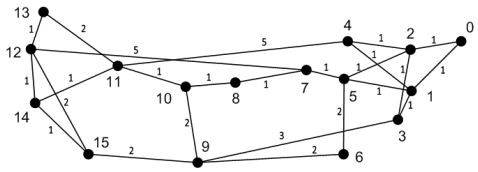
A. Simulation Setup

We tested the algorithm using two different network topologies. The simulations were carried out using a Python script running on a 2.70 GHz. The first is the NSF-Net graph, forming an approximate outline of the United States of America and placing nodes at major cities, including Chicago, New York, Atlanta, and Los Angeles. The second is a fully connected hexagon graph with a central node connecting the 6 vertices. The graphs of the same topologies differ from each other in network configuration and data center specifications. For example, Figure 1 features an NSF-Net graph, but with various link costs. Figure 2 features the same topology, however, each of its links has a cost of 1, and the graph denotes different data centers. Each of these data centers implements functions at a different price than Figure 1. Similarly, Figures 3-5 correspond to a fully connected hexagon graph with one data center B (in Figure 3), data centers B and C (in Figure 4) and data centers B, C and T (in Figure 5).

For each topology, we generated 4 requests $r \in R$ consisting of a group of functions $f \in F$ to be virtualized. Each one of these requests had a different source and destination node found across the topology that the route must traverse between. These routes are designated by Route 1-4 in the results (Figures 6 to 10).

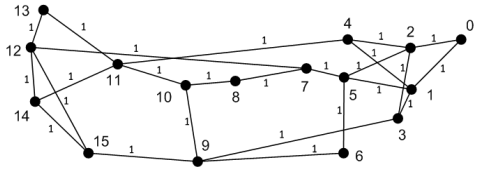
For every request within each topology, we only changed the source and the destination nodes (src_r, dst_r) of the route with respect to the various data centers. These costs of implementing the network function on the data centers were randomly distributed among a set of target values to provide a more reliable testing set. The evaluation metrics for the results are as follows: for each test case using the selected topology configuration (from Figures 1-5), the simulation compares the costs for the routes (Routes 1-4) as given by the proposed heuristic algorithm with that of the optimal ILP solution. Figures 6-10 depict the comparison of the routing cost, assignment cost and the overall cost, given by the sum of the routing and assignment cost, for the algorithmic and optimal ILP solution. The next section discusses the results in detail, in particular the gap between the proposed algorithm and the optimal ILP solution.

 ISBN: 978-1-61208-747-4



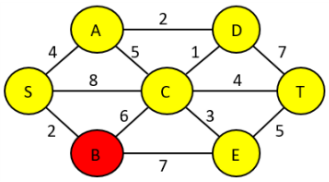
	Data Center 14	Data Center 10	Data Center 5	Data Center 0
Function1	5	1	10	14
Function2	9	7	6	10
Function3	3	14	9	3

Fig. 1. NSF 1 Network Graph



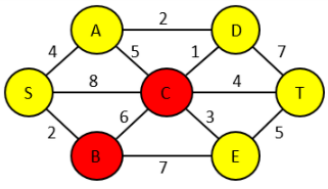
	Data Center 12	Data Center 8	Data Center 7	Data Center 5
Function1	12	8	7	5
Function2	6	5	1	15
Function3	6	10	4	7
Function4	6	3	6	8

Fig. 2. NSF 2 Network Graph



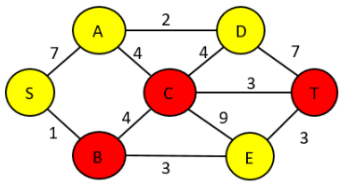
	Data Center B
Function1	3
Function2	6
Function3	1

Fig. 3. HEXAGON 1 Network Graph



	Data Center B	Data Center C
function1	3	6
function2	2	1
function3	4	4

Fig. 4. HEXAGON 2 Network Graph



	Data Center B	Data Center C	Data Center T
Function1	12	12	20
Function2	30	30	20
Function3	60	60	100
Function4	50	50	45

Fig. 5. HEXAGON 3 Network Graph

B. Simulation Results and Discussion

The first test involved the NSF-Net with routing costs randomly generated from 1 through 5. Data centers were placed at Nodes 12, 8, 7, and 5, with implementation costs given by Figure 1. The data centers were chosen to emulate central locations in the continental United States, allowing direct access to a larger set of nodes on the graph. The algorithm was able to compute a solution that equals the optimal total cost in Route 1, and was within 23% over the optimal cost in the remaining 3 routes.

In the second test using the NSF-Net, we set each route link to 1 and increased the weights of the data center cost, simulating an assignment cost-dominated scenario. Data centers were moved to 14, 10, 5, and 0 as seen by Figure 2. The algorithm matched the optimal net cost in two of the routes and was able to come close to reaching the optimal overall cost, within 22%.

The third test involved the fully connected hexagonal network, as shown in Figure 3. We designated one of the nodes on the periphery, B, as the data center and gave the assignment and routing costs an approximately equal weight. Since we assigned only 1 data center, the requests were able to meet the optimal assignment cost on each route. The overall algorithmic cost was an acceptable range of 20% within optimal cost.

The fourth test added another data center to node C, as shown in Figure 4. These tests were designed to add variation to the routing costs, as the request may route through multiple centers, incurring a different cost each time. Overall, as shown in Figure 9, the results of the algorithm come within 24% of the optimal case in each scenario.

The final test incorporated another data center at node T and increased the cost of assigning each function on a node, as shown in Figure 5. In addition, the cost of implementation for each function was increased. In this trial, as shown in Figure 10, the algorithm was able to come the closest to the optimal ILP solution, within 2.9%.

Overall, we find that the proposed heuristic algorithm produces solutions over the five different configurations that match the optimal ILP solutions within an acceptable range, while providing an efficient practical solution.

 ISBN: 978-1-61208-747-4

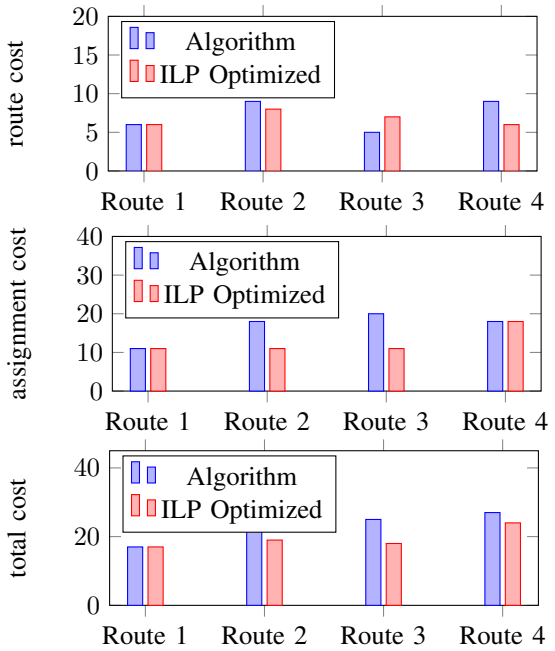


Fig. 6. NSF-Net 1 route, assignment and total cost comparison for Algorithm vs ILP Optimized

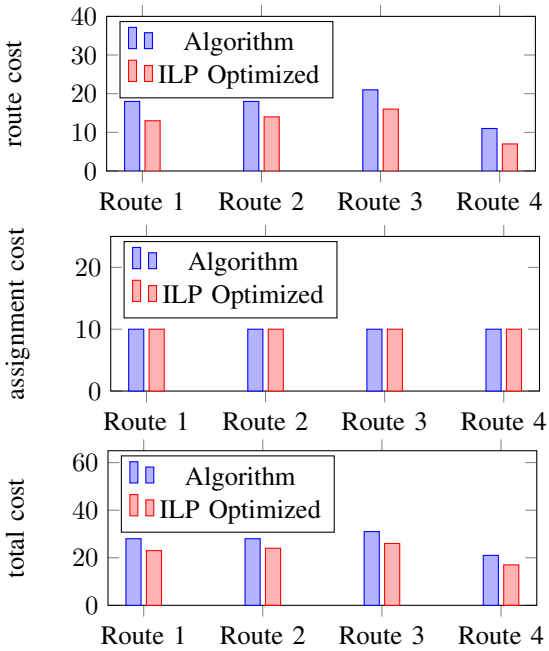


Fig. 8. HEXAGON 1 route, assignment and total cost comparison for Algorithm vs ILP Optimized

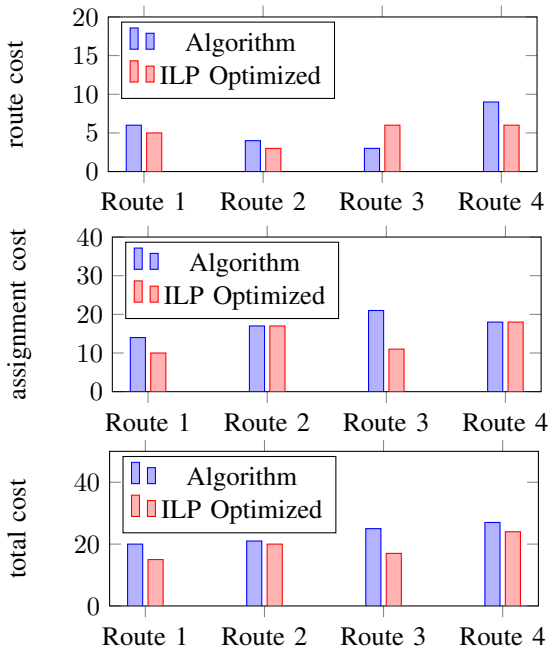


Fig. 7. NSF-Net 2 route, assignment and total cost comparison for Algorithm vs ILP Optimized

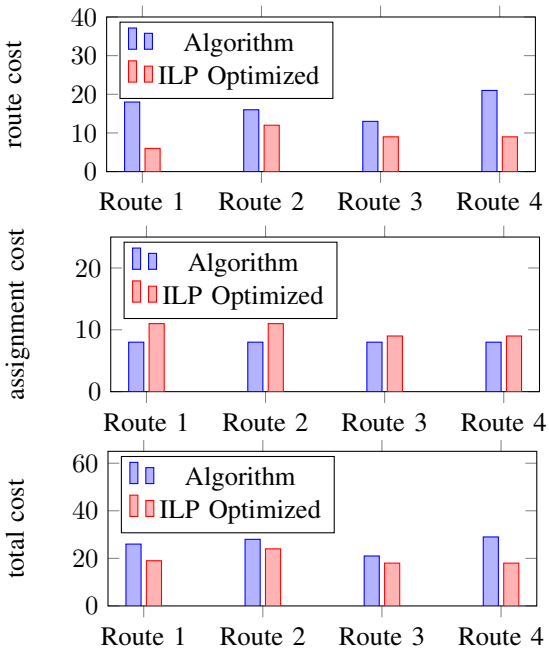


Fig. 9. HEXAGON 2 route, assignment and total cost comparison for Algorithm vs ILP Optimized

 ISBN: 978-1-61208-747-4

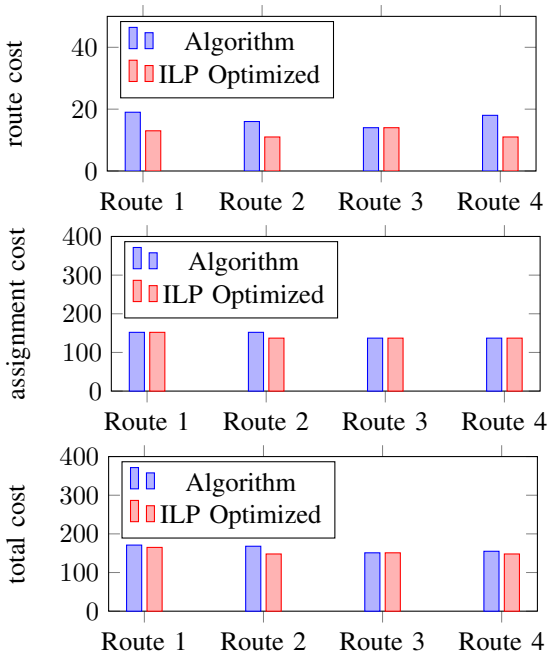


Fig. 10. HEXAGON 3 route, assignment and total cost comparison for Algorithm vs ILP Optimized

VI. Conclusions

This paper poses an optimization problem of service function assignment and shortest path for network function virtualization and proposes a solution that splits the problem into two separate optimization subproblems: shortest path and service function assignment. The algorithm uses the combination of a modified Dijkstra's shortest path algorithm and Greedy heuristic algorithm for network function assignment. Some of the key challenges lie in thoroughly validating the algorithm performance on larger networks. In future work, we propose to expand the scope of testing to validate the algorithm against larger network configurations, using more complex optimization packages such as Gurobi, CPLEX, etc.

References

[1] A. Blenk, A. Basta, W. Kellerer, and J. Zerwas, "Pairing SDN with Network Virtualization: The Network Hypervisor Placement Problem", *IEEE NFV-SDN*, November 2015, pp. 198-204.

[2] D. Amaya, Y. Sumi, S. Homma, T. Okugawa, and T. Tachibana, "VNF Placement with Optimization Problem Based on Data Throughput for Service Chaining", pp. 1-3. *IEEE CloudNet*, March 2018.

[3] M. Ghasem and L. Zhiquan, "Optimal Network Function Virtualization and Service Function Chaining: A Survey", *Chinese Journal of Electronics, Vol.27, No.4*, July 2018, pp. 704-717.

[4] D. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual Network Functions Placement and Routing Optimization", *IEEE CloudNet*, March 2015, pp. 171-177.

[5] J. Crichigno, D. Oliveira, M. Pourvali, N. Ghani, and D. Torres, "A Routing and Placement Scheme for Network Function Virtualization", *IEEE TSP*, July 2017, pp. 26-31.

[6] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions", *IEEE INFOCOM*, April 2015, pp. 1346-1354.

[7] M.C. Luizelli, L.R. Bayes, L.S. Buriol, M.P. Barcellos, and L.P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions", *IFIP/IEEE International Symposium on Integrated Network Management*, May 2015, pp. 98-106.

[8] H. Moens, and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions", *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 418-423.

[9] A. Gupta, M.F. Habib, P. Chowdhury, M. Tornatone, and B. Mukherjee, "Joint Virtual Network Function Placement and Routing of Traffic in Operator Networks", *Netsoft*, 2015, pp. 1-5.

[10] S. Skiena, The Algorithm Design Manual, *Springer-Verlag*, 1998.

[11] Coursera.com Algorithms Specialization: Graph Search, Shortest Paths, and Data Structures, *[Online] Available: coursera.com, [retrieved: 08, 2019].*

 ISBN: 978-1-61208-747-4