Genetic Algorithm for Time-Effective IoT Service Function Placement

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

Abstract—**Our paper focuses on the concept of IoT Service Function Chains (IoTSFC): a set of IoT Network Functions that must be allocated and implemented on IoT nodes in a specific order. Our problem is ILP and therefore NP-Hard, so an optimal solution cannot be found in polynomial time. Therefore, we attempt to devise a heuristic method to solve the problem at a lower time complexity. The paper develops a solution using a genetic algorithm that attempts to minimize the total processing time and incurred transmission delay time required to execute a group of network functions across IoT nodes. The genetic algorithm is run on a string denoting placement of the network functions and runs a set of genetic operators in order to work toward an optimal solution. Our experimental results have suggested that the algorithm can discover an optimal solution in an acceptable time frame.**

Keywords – Internet of Things, IoT Service Function Chaining, Minimax Problem, Genetic Algorithm, Natural Selection, Fitness.

I. Introduction

With the rise of network technologies in the last decade, the progress of the Internet of Things (IoT) has ramped up. Using various devices, such as sensors, remote monitors, etc., IoT networks can collect and process data on a massive scale. Services offered by an IoT device consist of various IoTSFC, and each IoTSFC itself contains a set of Network Functions (NFs). As opposed to less stringent network structures, IoTSFC functions must be implemented and executed in a specific order in order to carry out the appropriate service. With a rising amount of data, the efficiency of deployment becomes paramount. Traditional solutions involve function implementation on both hardware in the IoT gateway and in the cloud. However, this has proved unsuitable due to exorbitant costs and inflexibility. The introduction of Network Function Virtualization (NFV) has helped alleviate these concerns. Instead of implementation in the gateway, NFV allows for various NFs to be rendered through software on IoT nodes. This allows for programmability, as well as the required flexibility of network function assignment. Performance time can be improved by altering the site of implementation of a certain IoTSFC. However, high complexity and volume of these network functions can bring about challenges. The assignment of functions in IoTSFCs to viable IoT nodes is crucial to keep up with the increasing demands of today's age.

Section II delves into our contributions with this paper as compared to existing solutions. Section III establishes the mathematical model of the IoT problem. We specify the constraints and objective function that we look to minimize. Section IV introduces the proposed genetic algorithm we have devised, and it is further illustrated with a pseudo-code model. Section V depicts our experimental setup and testing of our proposed algorithm, featuring a Gantt chart to illustrate these results. Finally, Section VI contains the conclusion and final remarks.

II. Our Contributions

Our problem presents an approach to minimizing the combination of processing time and incurred transmission delays through the placement of IoT network functions across a set of IoT nodes. We utilize a genetic algorithm to discover a minimum solution, returning a full placement scheme for a set of network functions.

Ren et al. [1] delved into a new scheme for the placement of IoT service functions, attempting to deploy these functions on nodes that are as close to their data source as possible. Doing so allows them to minimize total resource costs and maximize system performance. Qu et al. [2] introduced the concept of delays in their paper regarding NFV. The authors explored two different types of delay, one of them being a transmission delay incurred by a switch in virtual machines. They looked to find a scheme to minimize the maximum time as well, treating the service function chain problem as a ”flexible job-shop scheduling problem”. The authors also utilized a genetic algorithm to discover a minimum solution. Gao et al. [3] developed a genetic algorithm to solve the job-shop problem, using various genetic operators to add diversity. Moghadam et al. [4] brought up the concept and execution of a POX crossover to be used in the offspring generation in a genetic algorithm.Kouah et al. [5] have developed an energy-aware optimization model to solve the placement of IoTSFC problem and proposed a genetic algorithm heuristic solution and an energy-agnostic algorithm, with the goal to avoid exhausting nodes with limited energy capacities and providing an optimal solution for energy consumption for a small network topology. Wang [6] has considered an IoTSFC placement problem from the perspective of minimizing the number of VNF instances implemented and proposed a genetic algorithm based solution.

This paper is unique in its inclusion of transmission delays in an IoTSFC placement scheme problem, utilizing a modified genetic algorithm in order to work toward a solution that minimizes the maximum time of implementation. While Ren et al. [1] attempt to minimize the distance from functions to a data source, we instead attempt to minimize the overall processing time and incurred transmission delay between network nodes. Wang [6] has considered an overall processing delay not to exceed a certain acceptable delay threshold, however has not accounted for transmission delay. Our problem formulation models closer to a real life IoT network as we take into account the varied transmission delays between network nodes. To our best knowledge, we are not aware of other research that has taken this into account, and this is a novel approach for solving IoTSFC placement problem. Our solution has been tested and is able to discover an optimal solution to the minimization in an acceptable time.

III. Mathematical Formulation of the Problem

A. Model

We represent a graph of IoT nodes $G = (V, E)$, where V is the set of nodes on the graph and E is the set of traversable edges connecting these nodes. We create a set of service requests $\{s_1, s_2, s_3, ..., s_N\}$, where each of these service requests consist of network functions $\{f_{i1}, f_{i2}, ..., f_{ik_i}\}$. Each node that implements these requests is denoted by $k \in V$. These network functions must be implemented in the exact order as specified in the IoTSFC.

We define $i \in \{1, 2, ..., N\}$ as an index of service requests. $j \in \{1, 2, ..., k_i\}$ is an index for each service request that denotes the network function in the necessary order. For a service request i, the processing time required to implement network function j on node k is defined as $t_{i,j,k}$. The time that this implementation begins is represented by $s_{i,j}$, and the time it ends is denoted by $e_{i,j}$. In addition, we include a transmission delay when a request implements network function j on node k and implements function $j+1$ on another node k', denoted by $d_{i,j,k,k'}$.

We denote $x_{i,j,k}$ to be a binary variable representing whether or not a network function j contained in service request i is implemented on node k. We denote $y_{i,j,k,k'}$ as another binary variable for service i, network function j is implemented on a node k while function $j+1$ is implemented on separate node k' (i.e., the node is switched from k to k').

B. Objective

We treat this problem as a "minimax" problem, attempting to minimize the combination of processing time and incurred transmission delays through the placement of IoT network functions across a set of IoT nodes. We must take into account both the time of implementation of each network function, as well as transmission delays incurred by switching nodes.

$$\min C$$

where C = max $\sum_i \sum_j (t_{i,j,k} \cdot x_{i,j,k} + d_{i,j,k,k'} \cdot y_{i,j,k,k'})$

C. Constraints

First, we must define binary variable $x_{i,j,k}$ to represent whether a network function j contained in service request i is implemented on node k.

$$x_{i,j,k} = \begin{cases} 1 \text{ if implemented} \\ 0 \text{ else} \end{cases} \tag{1}$$

Similarly, $y_{i,j,k,k'}$ as another binary variable that represents whether a service i, network function j is implemented on a node k while function $j+1$ is implemented on separate node k' (i.e., the node is switched from k to k').

$$y_{i,j,k,k'} = \begin{cases} 1 \text{ if (k, k') is traversed} \\ 0 \text{ else} \end{cases} \tag{2}$$

To prevent a network function j of service request i from being implemented on an underresourced node, we create a constraint governing the start time $s_{i,j}$ across all nodes $k \in V$. In addition, the function must be implemented after the previous function has ended, ensuring that the necessary order of functions is preserved.

$$s_{i,j} + \sum_k t_{i,j,k} \cdot x_{i,j,k} \leq e_{i,j}, \forall i, j \tag{3}$$

Furthermore, when switching nodes within a request, we must ensure that the next function is implemented following the incurred transmission delay, preserving the order of implementation.

$$e_{i,j} + \sum_k d_{i,j,k,k'} \cdot y_{i,j,k,k'} \leq s_{i,j+1}, \forall i, j \tag{4}$$

When switching a service i from function j to $j+1$ through node k to k', we assure the $j+1$ is implemented on k'.

$$\sum_k y_{i,j-1,k,k'} = x_{i,j,k'}, \forall i, j \tag{5}$$

Additionally, we create a constraint that makes sure we can only switch to one other node at each function in a request.

$$\sum_k^{\prime} y_{i,j,k,k'} \leq 1, \forall i, j, k \tag{6}$$

The final constraint governs the implementation of each service function i for a request j, allowing each to be implemented only once on a node.

$$\sum_k x_{i,j,k} = 1, \forall i, j \tag{7}$$

Our problem is ILP and therefore NP-Hard, so an optimal solution cannot be found in polynomial time [6]. Therefore, we attempt to devise a heuristic method to solve the problem at a lower time complexity.

IV. Genetic Algorithm for IoT Function Placement

A. Proposed Solution Setup

In this paper, we propose a genetic algorithm that will allow us to schedule the implementation of a set of network functions onto IoT nodes. We attempt to minimize the combined processing time and incurred delay times of IoT service function chains assigned across IoT nodes.

In order to do so, we designate two strings [3] representing the function placement to be utilized by the algorithm. The first is the Operation Selection (OS) string, listing a set of functions given in the order they are meant to be implemented in. A service request i with j network functions, for example, is listed j times in the total string in various positions. The first network function will be listed as the first appearance, the second network function the second, and so on. The nth listing of a service request is the nth network function it contains, for example. This string cannot be edited, and is given as an input into the algorithm.

The second string, known as the Machine Selection (MS) string, outlines the nodes that each of the functions will be placed on. It has the same length as the OS string, however, it is split up into its i components, one for each service request. The jth element of the ith component represents the node on which jth function of service function i is to be implemented, as defined by the OS string. Each smaller component, denoting a service request, represents a chromosome. Each element, containing the node of implementation, is a gene. A solution is represented by placements dictated by an MS string.

To decode the MS string and return a set of $x_{i,j,k}$ and $y_{i,j,k,k'}$ values, we iterate through the string. If the value of the jth network function in the ith service request chromosome is equal to some value k, we set $x_{i,j,k} = 1$. For all network functions in a gene, if the value is a different value k' from the previous, indicating a different node used to implement, we set $y_{i,j,k,k'} = 1$.

For example, we consider a set of 3 jobs containing 2 functions each. These $f \in f_{1,1}, f1, 2, ..., f_{3,1}, f_{3,2}$ may be implemented on one of 3 virtual nodes.

We randomly designate an OS String, in this case, to 131223, denoting the order of function placement. The initial value is the first repetition of 1, so we implement $f_{1,1}$ first. The second function to implement is $f_{3,1}$, third is $f_{1,2}$, and so on. Similarly, we create a MS String of 213122. $f_{1,1}$ is implemented on Node 2, $f_{1,2}$ on Node 1, $f_{2,1}$ on Node 3, and so forth.

B. Proposed Genetic Algorithm

We begin by setting an arbitrary population size to store the MS strings. We define a simple fitness function as the maximum time taken to implement all the functions on the nodes denoted by the string. We generate two MS strings randomly, and unpartitioned to begin. To generate a population, it must satisfy each of the constraints listed previously. We select a node at random for each network function of each service request. Going in the order dictated by the OS string, we create a suitable starting time for each function after the previous function has been implemented, using the idle times available on each node. If there is enough time in an idle slot, we may add the next function in it. If not, it will be implemented at the end. These strings are stored in a sorted queue of length equal to the previously selected population size.

Inside the loop, we begin by evaluating the fitness of our population. This fitness test is done by calculating the time required for implementation, and a lower time entails a higher fitness score. We also define termination criteria; in this paper, the algorithm terminates when we reach the maximum generation or the maximum step with no significant improvement. If the termination criteria are satisfied, we exit and return the MS string that provides the optimal time at the beginning of the queue.

If not, we select the two MS strings that provide the lowest maximum time to implement all functions in the OS string. These strings are crossed over to create two new offspring. We utilize a POX crossover [4] that will allow us to remain within the given constraints. First, the service requests contained in the string are partitioned at random into two groups. An element in the first parent string that is sectioned into the first group is placed in the same position in the first offspring, then removed from the first parent string. Meanwhile, elements in the second parent string that are in the second group occupy the same position in the second offspring and are removed from the parent. Elements that remain in the first parent string are placed in that order in the remaining spots of the second offspring, and vice versa.

We also introduce simple swapping mutations to the offspring after crossover. Two elements chosen at random in the offspring strings may be swapped, adding another layer of randomness to the selection process. After the offspring are completed, we evaluate their fitness as well. If any offspring are deemed to be more fit than any elements in the population, we add these offspring to the population and remove the least fit members of the population in order to maintain the constant population size.

After the loop is completed, the algorithm will return the updated values of $x_{i,j,k}$ and $y_{i,j,k,k'}$ for all i, j, k.

V. Experimental Results and Discussion

We have tested the proposed algorithm using the network graph depicted in Fig. 1. The topology features six IoT nodes, with links connecting each node to its adjacent neighbors. We also removed the link from Node 1 to 3, so a service request may not switch between the two. Along each connected node pair, we labeled the transmission delay that would be incurred by switching across the two nodes. As suggested earlier, this closely models a real IoT network as we take into account the varied transmission delays between network nodes.

For this topology, as shown in Fig. 2, we created a set of four requests with a random amount of network functions contained in each. For example, Service Request 1 contains both f_{11} and f_{12}, while Service Request 3 contains three functions f_{31}, f_{32}, and f_{33}. We then defined a random implementation time for each network function on the six nodes given by the topology. Some of the functions cannot be implemented on all nodes; for example, f_{31} can only be implemented on Nodes 3 and 6. During simulations, we set the processing time

Algorithm 1 Genetic Algorithm for IoTSFC Placement

```
  Input: $G(V, E)$; $t_{i,j,k}$; $d_{i,j+1,k,k'}$; OS string
  Output: $x_{i,j,k}, y_{i,j,k,k'}$
population = empty sorted queue of length $popsize$
MS1, MS2 = randomly generated MS strings
population = population ∪ MS1, MS2
while TRUE do
    evaluate fitness of all elements in population
    if termination criteria are met then
        optimalString = MS string with highest fitness score in population
        for all element in optimalString do
            update $x_{i,j,k}, y_{i,j,k,k'}$ using element
        end for
    end if
    P1, P2 = two MS strings from population with the highest fitness, implementing network functions in lowest times
    O1, O2 = POX crossover on P1, P2
    randomly mutate O1, O2
    if fitness of O1 $\geq$ lowest fitness in population then
        add O1 to population
        remove least fit element of population
    end if
    if fitness of O2 $\geq$ lowest fitness in population then
        add O2 to population
        remove least fit element of population
    end if
end while
return $x_{i,j,k}, y_{i,j,k,k'} \forall i, j, k$
```

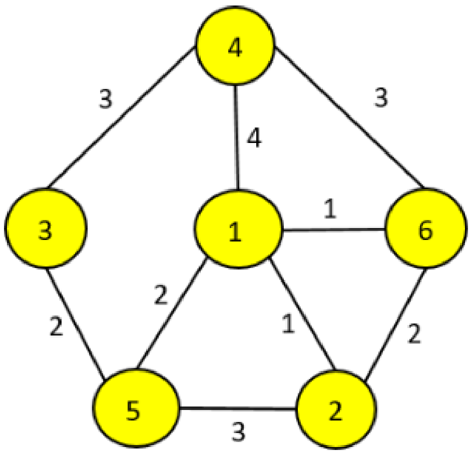
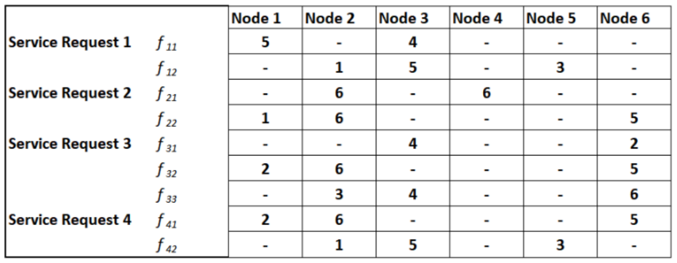


Fig. 1. 6 IoT-node topology

for any unimplementable function on a certain node to be a very large integer, preventing the cost-reducing algorithm from implementing there.

As shown in Fig. 3, our test was able to execute each network function contained in each service request in a total time of 7 units. The final and optimal string requires five node switches, incurring delays each time. Node 1 implements



		Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Service Request 1	f_{11}	5	-	4	-	-	-
	f_{12}	-	1	5	-	3	-
Service Request 2	f_{21}	-	6	-	6	-	-
	f_{22}	1	6	-	-	-	5
Service Request 3	f_{31}	-	-	4	-	-	2
	f_{32}	2	6	-	-	-	5
	f_{33}	-	3	4	-	-	6
Service Request 4	f_{41}	2	6	-	-	-	5
	f_{42}	-	1	5	-	3	-

Fig. 2. Set of 4 service requests with NF implementation times in each node

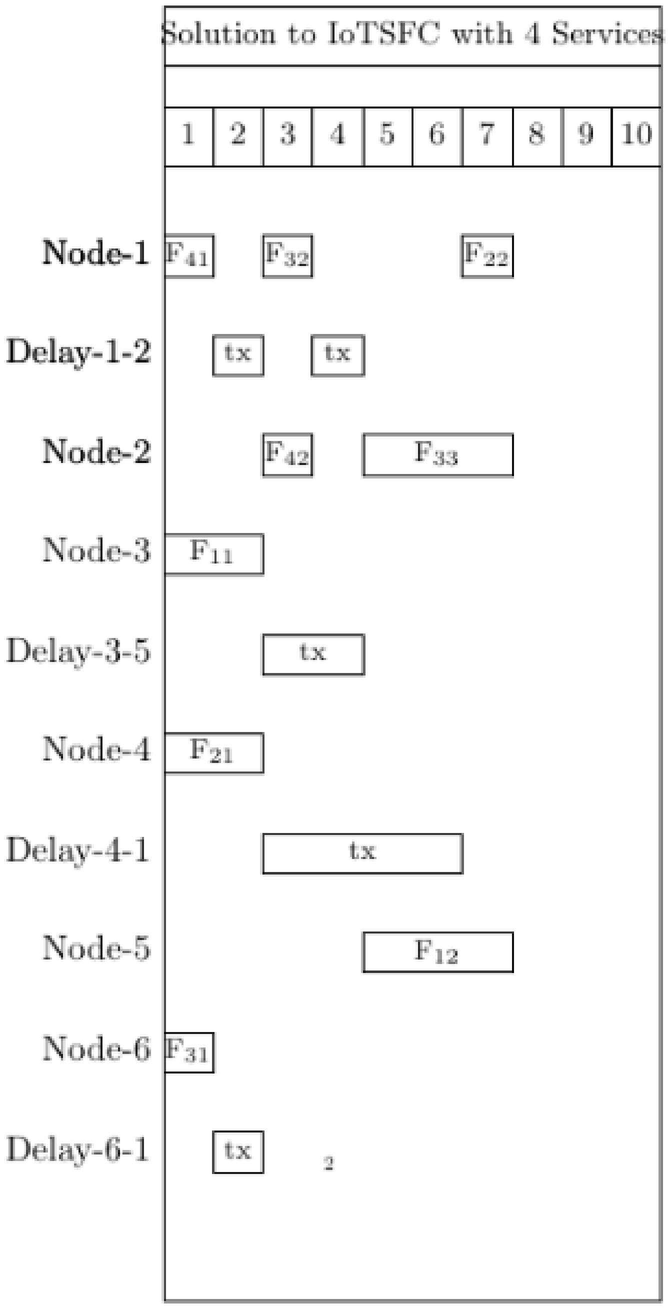


Fig. 3. GA Algorithm solution for IoTSFC Placement for 4 service requests on 6 IoT-node topology

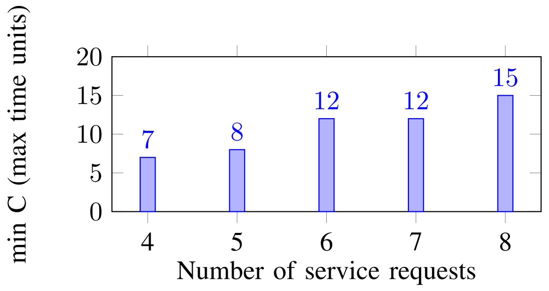


Fig. 4. Transmission Delay = 1

the most service functions, with three, while Nodes 3, 4, 5, and 6 are only tasked with implementing 1 function each. The algorithm took 15.55 seconds to run, creating additional generations until this solution was reached.

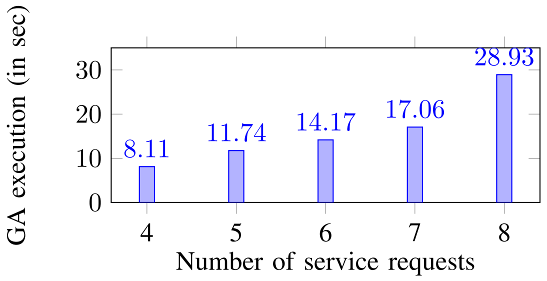
We further tested our proposed algorithm using the same network topology as Fig 1. However, instead of using a variable set of transmission delays as shown in Fig 1, we used a constant delay when switching between functions and across all connected links. We tested the algorithm with two different network configurations - 1) by setting the transmission delay to 1 time unit, and 2) increased the transmission delay to 2 time units.

For the two network configurations, we generated a set of service requests containing various network functions. For each request, the number of functions contained was randomized, as well as the implementation costs on each of the six nodes. Each function may not necessarily be implemented on all nodes as well.

As shown in Figure 4, we varied the number of service requests from four to eight for the network configuration with transmission delay = 1. Our algorithm was able to discover the optimal placement of the network functions across the given topology to minimize both processing costs and the transmission delays incurred when switching nodes. For example, in Figure 4, in the number of service requests = 4, the algorithm found the optimal placement (processing and transmission delay time) of 7 time units, and the corresponding GA algorithm execution time was 8.11 seconds. As the number of service requests was increased to 8, the algorithm found the optimal placement of 15 time units, with the execution time of 28.93 seconds. Similarly Figure 5 shows the algorithm optimal placement solution with the corresponding algorithm execution time for the network configuration with transmission delay of 2 seconds.

VI. Conclusions

We attempted to minimize the combination of processing time and incurred transmission delay of the placement of IoT network functions across a set of nodes. Since this problem falls under the classification of ILP problems, it is NP-Hard and therefore cannot be solved in a polynomial time. We utilized a genetic algorithm to discover a solution to the problem at a lower complexity. The algorithm takes into



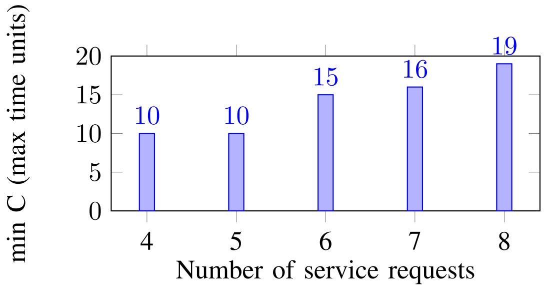
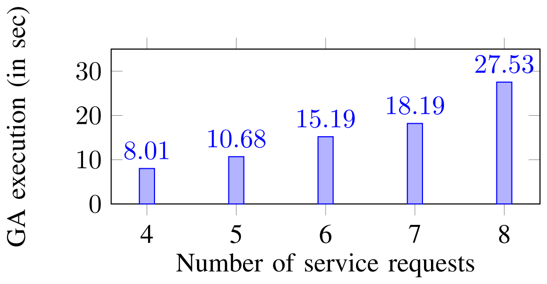


Fig. 5. Transmission Delay = 2



account both execution time and transmission delays incurred by a request switching nodes and experimental results over IoT network configurations suggest that the algorithm provide optimal placement in an acceptable time frame.

References

[1] W. Ren, Y. Sun, H. Luo, and M. Obaidat, A New Scheme for IoT Service Function Chains Orchestration in SDN-IoT Network Systems, *IEEE Systems Journal*, pp. 1-12, July 2019.

[2] L. Qu, C. Assi, and K. Shaban, Delay Aware Scheduling and Resource Optimization with Network Function Virtualization, *IEEE Transactions on Communications*, pp. 3746-3758, Sept. 2018.

[3] L. Gao and X. Li, An Effective Hybrid Genetic Algorithm and Tabu Search for Flexible Job Shop Scheduling Problem, *International Journal Production Economics*, pp. 93-110, Dec. 2015.

[4] A.M. Moghadam, K. Wong, and H. Piroozfard, An Efficient Genetic Algorithm For Flexible Job-shop Scheduling Problem, *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 1409-1413, Dec. 2014.

[5] Xin-Gang Wang, An Effective Solution Based on Genetic Algorithm for Virtual Network Functions Placement, *ICEIT*, pp. 21-31, 2017.

[6] R. Kouah, A. Alleg, A. Laraba, and T. Ahmed, Energy-aware placement for IoT-Service Function Chain, *IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, pp. 1-7, 2018.

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