Algorithms for VNF Placement and Chaining in Multi-Access Edge Computing considering an IoT setting

Lukas Graber
Technical University of Munich (TUM)
Munich, Germany
lukas.graber@tum.de

ABSTRACT

Recent technologies like 5G and the Internet of Things (IoT) require more dynamic and flexible services as currently provided by most infrastructures. This introduces new challenges for service providers and network operators. As a countermeasure, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) were introduced. NFV provides network functions as virtualized instances, also referred to as Virtual Network Functions (VNF). On a similar note, Multi-Access Edge Computing (MEC) moves computing power to the network edge in close proximity to the end user. Service requests within the network consist of different VNFs chained together, building a service chain (SC). The placement of these VNFs on nodes in the network in an effective and meaningful way is termed the VNF placement and chaining problem. As this problem is known to be NP-hard, network operators are interested in allocation schemes that exhibit acceptable performance and provide more flexible and dynamic services. For that purpose, this paper constructs a rather detailed system model for a network infrastructure that also includes IoT sensor networks. The resulting objective function for the VNF placement and chaining problem optimizes for latency and availability. Different algorithms and heuristics are presented that approximate the optimal solution for the VNF optimization problem. This paper serves as a guide on how to construct a VNF placement and chaining algorithm with certain properties (e.g. latency, availability, cost) in mind. The resulting VNF allocation scheme utilizes the network edge and integrates IoT features into the network infrastructure. The presented construction process can be streamlined and adapted by network operators for real-world scenarios that should also respect IoT features.

KEYWORDS

Multi-Access Edge Computing (MEC), VNF Placement and Chaining, Edge Clouds, Internet of Things (IoT)

1 INTRODUCTION

Over the last decade, the world has seen a surge in the number of mobile devices. The introduction of new technologies like 5G, the Internet of Things (IoT) and Industry 4.0 poses new challenges, like low latency and high reliability services, for the existing infrastructure and service models [13]. In traditional telecommunications, network operators deployed different services on proprietary hardware appliances [3][4][12]. Due to the requirements of more flexible and diverse services in recent years, this approach has become increasingly outdated. Not only the deployment of network resources, but also the maintenance of these hardware appliances drive up the Operational Expenditure (OPEX), as well as the Capital Expenditure (CAPEX) for Telecommunication Service Providers (TSPs) [3][4]. Therefore, service providers are looking for new ways to orchestrate and build more flexible networks [12].

To tackle these challenges, two new concepts were introduced: Software-Defined Networking (SDN) and Network Functions Virtualization (NFV)[4]. SDN separates the network's control and data plane and thereby makes the network more flexible and easier to manage for network operators[3]. This allows for programmable and centrally managed networks[4]. NFV on the other hand decouples network functions (NFs) from the hardware plane by using virtualization technology to run those services as virtual network functions (VNFs) on standard servers [3][5][12][14]. Those VNFs can be moved, instantiated or redeployed at different locations in the network without the cost of installing new hardware appliances [12]. As those functions run inside virtual machines (VMs), they introduce service scalability, i.e. vertical/horizontal scaling, into the network. It should be mentioned that these concepts are not limited to network functions, but can be applied to more differentiated services like IoT applications. SDN in combination with NFV enables service providers to build cost-effective, flexible and scalable networks and services. A service is usually composed of more than one network function that can be located at different physical locations in the network. Thus, in order to provide a requested service, its traffic must be routed through the chain of associated VNFs in a pre-defined order, also referred to as a Service Chain (SC).[3][14]
As NFV on its own can not provide for the low latency requirements defined by the Service Level Agreements (SLAs) between the service provider and the customer[7], infrastructure and service location must also be considered. Edge Computing (EC) is one such concept, which differentiates the network into edge and core infrastructure and allows for the deployment of services at the edge in close proximity to the end user [13]. This reduces access latency compared to central cloud access. The European Telecommunication Standards Institute (ETSI) coined the term Multi-Access Edge Computing (MEC), formerly known as Mobile Edge Computing[1], to describe the technology's standard architecture[2]. However, due to the limitation of available resources at the network edge, one of the key challenges of MEC is the ability to locate latency sensitive services in the appropriate edge compute nodes, according to the specific demand for each of the services and the relevant latency constraints [5]. As services are usually composed of multiple VNFs

[1] https://www.sdxcentral.com/edge/definitions/mobile-edge-computing-vs-multi-access-edge-computing/
[2] https://www.sdxcentral.com/edge/definitions/what-multi-access-edge-computing-mec/

that are allocated at different physical locations, the performance of the whole chain is highly VNF-location sensitive [14]. This resource allocation is generally called the VNF placement and chaining problem. Network operators must find a way to place the VNFs at the right location, when they are needed. This problem becomes even more challenging when dynamic situations, e.g. online arrival of network services or changing sequence of VNFs, are considered[12]. With integration of new technologies like the IoT, more dynamic scenarios are introduced in SDN/NFV environments. Therefore, TSPs must consider not only the right location, but also the right timing in their operation and management plan. As defined by ETSI, the NFV Management and Orchestration (MANO) is responsible for instantiation, placement and chaining of VNFs [4]. The NFV orchestration process is designed to automate the network service deployment over the virtualized infrastructure. This can be achieved by implementing a central controller via SDN to deploy services across the network.

The number of IoT connected devices is predicted to increase drastically over the coming years[14]. The operation of sensor networks in the IoT can greatly benefit from the emergence of SDN/NFV in MEC networks. In the IoT, the network architecture can be further differentiated into the IoT layer, e.g. sensor networks, the network edge and the core cloud. IoT devices, e.g. sensors, generate vast amounts of data that is routed through gateways to computational resources in the edge and cloud layer [7]. Many of these IoT-based applications that rely on MEC show high-mobility features, e.g. smart cars, that need to be respected by network operators [14]. Thus, allocating a static IoT edge may not be reasonable, considering the fact that the optimality of the edge allocation scheme varies based on ongoing events [10][14]. Mobility-unaware VNF placement in the IoT can therefore lead to performance degradation for IoT applications [14]. To guarantee latency constraints for mobile IoT devices, the placement selection must be updated frequently by considering current events. Thus, also sensor mobility must be taken in account in the placement and orchestration scheme of network operators.

The VNF placement and chaining problem can be understood as an optimization problem which optimizes for specific network properties, e.g. latency, availability and network cost. A formal representation of the network infrastructure and properties is needed in order to formulate the optimization goals as an objective function that is subject to constraints that are specific to the use case. As the number of compute nodes increases, the problem of resource allocation quickly becomes combinatorial in its nature [3]. The optimization for the IoT introduces additional elements, which increase the system complexity and create new constraints for the VNF placement and chaining problem. Optimization problems can be formulated as (Mixed) Integer Linear Programs (MILPs). MILPs can be solved by powerful mathematical solvers, e.g. the CPLEX software package that utilizes optimization algorithms like Simplex Algorithm [3][12]. But since the VNF placement and chaining problem is known to be NP-hard [3][4][5][10][13], those programs do not scale well. To this end, multiple approaches were introduced that attempt to solve the problem more efficiently[9]. One of these approaches is the derivation of algorithms and heuristics based on the original MILP formulation. These heuristics should generate near-optimal solutions in a computationally less expensive way.

The contribution of this paper is two-fold: 1) A detailed system model is provided that yields an objective function, optimizing for availability and latency. The problem formulation further considers deployment cost and integrates IoT features. 2) To tackle the computational intractability of the optimization problem, different algorithms are proposed that also respect dynamic features for IoT services and scalability requirements of network operators. The schemes provided by this survey are heavily based on the existing work presented in section 2 and need to be adapted for real-world implementations.

1.1 Overall Architecture

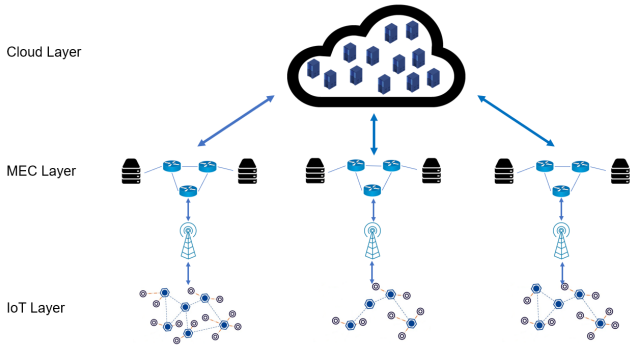

Figure 1: IoT scenario - The network is separated into three different layers: 1) the IoT layer, 2) the MEC layer and 3) the core cloud

The overall architecture of the system is based on [10] for the most part. The network can be differentiated into three different layers as can be seen in figure 1: 1) the IoT layer, 2) the MEC layer and 3) the core cloud. If a service request is generated, the network flow will be routed through the VNFs belonging to the associated service chain. It should be noted, that the presented architecture model is not only limited to network functions. In principle, any application that can be virtualized can be deployed as VMs on the data centers in the cloud and network edge.

The IoT layer is composed of sensor networks that are connected to the MEC layer through gateways. A sensor can only be connected to exactly one gateway through its interfaces, e.g. Bluetooth or WiFi. Each IoT device in the network can either collect data as a sensor, e.g. a temperature sensor, and pass it to the gateway or act as an actuator, e.g. a door lock, and receive control messages from the gateway in order to perform some action. A gateway on the other hand can be connected to multiple edge clouds through its multiple interfaces (i.e. cellular, LoRA).[10]
The sensor networks generate vast amounts of data that will be aggregated and partly pre-processed by the gateways. Allocated VNFs in the network then receive sensor information via the gateway and send control messages back to the IoT devices.[10] The intermediate MEC layer is crucial for IoT scenarios, as the processing at

the edge reduces the amount of data that will be forwarded into the core cloud. This diminishes the likelihood of network congestion.

Services can be differentiated into two different types: delay sensitive services and computational intensive services. Delay sensitive services, e.g. firewalls, should deployed at the network edge, in close proximity to the end user as smaller physical distance generally means lower latency. Computational intensive services, e.g. Deep Packet Inspection, are rather processed in the data centers of the cloud, as these locations provide more resources[7]. Thus, VNFs may come with location constraints. VNFs also have the ability to modify the rate of data, e.g. a firewall. It is possible that for a particular service chain, functions at the edge as well as the cloud layer will be visited in order to service an associated service request. The flow along this path does not have to be a straight line, but can also be bifurcated [7]. As VNFs are executed inside virtual machines (VMs), concepts of VM scaling can be applied: Vertical Scaling means allocating more resources to the underlying VM of a specific VNF. This is usually accompanied by a certain down time of the VM as it needs to be shut down for the scaling to take effect. Horizontal Scaling distributes the work load onto several VNF instances of the same VNF type. These instances do not necessarily need to be located at the same physical node. No down time is expected for horizontal scaling.[3] A network operator can apply VNF scaling in order to adapt the service allocation scheme to the current workload.
As an actual implementation of this architecture, I refer to the proposed model in [10], which is partly based on the MANO reference architecture. As the main focus of this paper is the algorithm side of VNF placement and due to lack of space, specific information on the actual deployment and implementation of the architecture is omitted in this paper.

1.2 Exemplary Use Cases

There are multiple scenarios where MEC and the IoT can be combined. Two sample IoT scenarios should be pointed out: a static and a dynamic scenario.

1.2.1 Static Scenario. The static use case describes a security surveillance system at home as can be seen in [10]. The system uses data generated by a motion sensor, supported by additional data from a camera sensor. The evaluation and decision making is delegated to service functions at the edge. The service chain is triggered by the motion sensor and a respective VNF will decide on consulting camera data for further analysis. Coupled with face recognition functionality, a potential housebreaking will be detected and endpoints, e.g. smartphones, will be informed along the service chain.

1.2.2 Dynamic Scenario. A dynamic scenario is outlined in [1]. A traffic control system for smart cities is presented that ensures smooth operation of day to day traffic as well as managing emergency vehicle movement with minimal delay. At every traffic junction, there are sensor-based devices located, communicating current local traffic data to a nearby IoT gateway. Multiple gateways can be deployed on a MEC server that also hosts a Local Traffic Analytics Engine as a service function. This function takes regular input of traffic lights provided by the IoT gateways and applies Machine Learning algorithms to generate traffic patterns. This information can then be used by an IoT gateway to determine the operating sequence of traffic lights. Decision making is supported by a Centralized Traffic controller in the cloud which connects to every traffic light individually. There are also hospital servers connected to the infrastructure over the cloud. These hospital servers contain databases of all the registered patients and their medical history. A mobile application is provided for a patient's device, which can detect a patient's medical status, e.g. fall detection or heart status. In case of an emergency, such a device calls the hospital and transmits the patient's unique id and current location. The hospital will then find the nearest ambulance and provide patient data for preparation. Furthermore, the hospital servers will send data of the selected ambulance to the traffic controller in order for it to find the fastest route and optimize traffic along the way.

1.3 Outline

The rest of the paper is structured as follows: Related work is discussed in Section 2. Section 3 introduces the system model and in Section 4 the VNF placement and chaining problem will be formalized and integrated into an objective function, respecting identified key properties, e.g. latency, availability and cost. Section 5 presents different types of algorithms and meta-heuristics that can be used to simplify and optimize the multi-dimensionality of the constructed objective function. Finally, Section 6 draws the conclusion.

2 RELATED WORK

As the problem of VNF placement, routing and chaining is known to be NP-hard [4], the topic has gained considerable attention from research. One basic approach to the problem of VNF placement and routing is described in [4]. The main contribution of this paper is a compact formulation of the problem using the MILP model to reduce network latency. The suggested solution does not differentiate between the MEC layer and the core cloud. Furthermore, all service requests are already admitted into the network, thereby ignoring the dynamic nature of real-world implementations. As also mentioned in the paper, their solution falls flat when encountering a large scale scenario. To overcome this scalability challenge, it is suggested to incorporate a greedy algorithm or a heuristic together with their proposed objective function.

A more realistic approach is suggested in [7]. The starting point of their research is a system model, which distinguishes between the MEC layer and the core cloud. The main focus of their work is put on minimizing communication delay and the overall deployment cost. The entire system model looks at a vast number of different constraints that should be taken into account by the resulting objective function. Nodes in the network model are differentiated into servers and routers. The proposed architecture is similar to the one described in this paper, differentiating between MEC and cloud layer. The work load is characterized by a set of SCs that are comprised of multiple VNFs, being executed in VMs at different physical locations. Furthermore, it is specified that some VNFs may come with location constraints, meaning some VNFs can either be processed in the core cloud, e.g. video optimization or deep packet inspection, others only at the edge, e.g. NAT or firewall. The

network model is constructed in such a way, that topologies with bifurcated paths are allowed. Moreover, VNFs that actively modify the rate of traffic, e.g. firewalls or video rendering services, are taken into account in their final objective function. As the minimization of end-to-end latency is a main focus of the paper, four different types of communication delays are identified: 1) propagation delay, 2) transmission delay, 3) processing delay and 4) queuing delay. The behaviour of different users over time is modeled by a Poisson Point Process. To demonstrate this behaviour, the processing of the communicating devices is described with the M/M/1 queueing model. With all these constraints taken into account, an objective function is constructed which minimizes both deployment cost as well as overall delay. Due to scalability issues with the proposed MILP formulation, they introduced an additional meta-heuristic Tabu Search (TS) Algorithm, which avoids being trapped in local optima. They appear to achieve significant improvements in terms of performance compared to the CPLEX implementation of their MILP formulation.

MEC is tightly coupled with developments in 5G, such as providing the needed infrastructure to enable ultra-Reliable Low-Latency Communications (uRLLC) services. Focusing on especially these services, [13] provides a system model that considers two conflicting objectives, namely minimizing access latency and maximizing service availability. To solve the computational intractability of the MILP formulation, they propose a Grouping Genetic Algorithm (GGA). To unite the latency as well as the availability constraint, a specific policy is integrated into their objective function to provide the option to tailor to a certain use case. In their experiments, the solutions provided by the proposed GGA are close to the optimal solution, but computed faster compared to the ones of the CPLEX implementation of their system model.

A more theoretical approach is provided by [5]. The paper tackles the challenge of Virtual Network Functions Placement and Assignment Problem (VNFPAP) in MEC. To this end, two new algorithms with guaranteed performance bounds are introduced. The objective is to find an allocation of network functions to infrastructure nodes, as well as an assignment of clients to those nodes, such that latency demands are preserved and the number of satisfied customers is maximized. One of the algorithms solves the decision problem in polynomial-time and determines whether a solution exists that satisfies all clients. The second algorithm is a deterministic $(1 - \frac{1}{e})$-approximation algorithm that finds the maximum set of clients that can be satisfied (MAX VNFPAP). As was shown in the experiments, the proposed solution performs much better than current smart random heuristics. Despite presenting an algorithm with a guaranteed performance bound, the outlined proposal is missing essential components that should be considered for a real-world IoT scenario. Latency is only described as the distance between the client and the infrastructure node. Furthermore, the infrastructure is not further differentiated into separate layers, e.g. MEC layer and core cloud. For the most part, the paper limits the number of functions that can be hosted on an infrastructure node to a maximum of one. This will probably never be the case in an actual implementation. As a result, the authors of the paper suggest a slight modification to their original algorithm to take account of multiple functions being placed at the same physical location. In the end, the paper provides a computationally bound framework for placing VNFs in the network that should be extended for the IoT.

Another important feature that needs to be taken into account is the dynamic nature of NFV and its importance for 5G and MEC. The future algorithms for VNF placement need to be tailored to adapt to a rapidly changing state of the network. [12] is one of the papers that elaborates on this notion and builds a network model and algorithm that at its core is designed to be deployed in dynamic scenarios. Different dynamic characteristics of network services are identified and integrated into their system model. The suggested algorithm DAFT is a primal-dual based framework which has a proven $(1 - \frac{1}{e})$ competitive ratio. In their simulations, it is shown that their algorithm has superior performance over several other algorithms. As [12] is mainly focused on optimizing for dynamic environments, which certainly has its importance for the IoT, other relevant properties such as latency and/or availability are neglected.

IoT mobile services, e.g. technologies in smart cars and high-speed E-health services, must integrate dynamic features for a useful decision on how to place those services at what location and time under what constraints. Focusing on the dynamic part of IoT mobile services, [14] presents a proactive Edge-Chaining (PEC) scheme formulated as an ILP framework, which handles service requests in a batch-like manner. The derived probability-prior PEC (PPEC) algorithm reduces the effect of dimensionality of the original ILP by taking end-device mobility patterns and possible destinations of IoT devices into account.

A more distributed approach was undertaken by [3]. In their model, different VNFs of a service request can be located at different edge clouds and a service provider can assign weights, e.g. link bandwidth or link delay, to links in the network. As a service request comprises of several VNFs, a path through these VNFs can be constructed. By using a multiple shortest path algorithm, multiple candidates can be calculated. The best candidate, that actually satisfies the resource constraints, is selected. A solution for the VNF placement and routing problem was suggested as a non-linear integer mathematical program. Linearization techniques are applied in order to utilize powerful mathematical solvers. In addition to that, [3] considers the scalability of VMs. Two algorithms for horizontal, as well as vertical scaling are proposed.

Besides VM scaling, it might also be of interest to respect network size in the VNF placement problem. [10] considers the VNF placement problem specifically tailored to an IoT setting. A non-convex integer programming model is formulated and two algorithms for small- and large-scale networks are proposed: 1) A Markov approximation algorithm introducing multi-start and batching techniques (MBMAP) and 2) A node ranking-based heuristic (NRP). In their experiments, they were able to show that the proposed algorithms can drastically reduce system cost.

In this work, I attempt to combine valuable parts of the aforementioned papers to propose a solution for the VNF placement and chaining problem, adjusted to the IoT. Besides minimizing latency, deployment cost and maximizing availability, the dynamic features of IoT networks should also be considered. The formal representation of the network infrastructure should account for scalability to different network sizes and provide the option for a network operator to scale the VMs to its needs. The resulting objective function and constraints will build the foundation for constructing algorithms for the VNF placement and chaining problem. To this end, the next section introduces a formal representation of the system model that will be used to express the identified properties in later sections.

3 SYSTEM MODEL

The network infrastructure can be described as a directed weighted graph $G = (V, E)$, V being the nodes and E the edges of the network. The system can be further differentiated into $V = I \cup G = M \cup C \cup Q \cup G$. The IoT sensor networks are connected to the service infrastructure $I = N \cup Q$ through gateways G. IoT gateways G forward aggregated data from connected sensors, potentially preprocessed, to VNFs through the network link between the gateway $g \in G$ and the cloud N. I is composed of computing nodes N, e.g. servers, and forwarding nodes Q, e.g. routers. VNFs are only deployed at $N = M \cup C$, where M are the MEC nodes and C describe the cloud data centers (DCs) in the core network. Some VNFs may come with location-constraints and can only be executed either at the edge or in the core cloud. The set $T \subset N$ is defined to describe these location constraints, such that node $n \in T$ can only host a specific type of VNF.

Multiple service chains $S = (S^1, ..., S^s)$ can be deployed in the system at the same time. As already mentioned, a service chain $S^i = (v_1^i, ..., v_h^i)$ is composed of multiple VNFs v_k^i. A VNF $v_k^i = (v_{k1}^i, ..., v_{ko}^i)$ can be deployed as multiple instances v_{kj}^i (e.g. containers or VMs) on different physical servers N to allow for horizontal scaling. At the same time, a physical node $n \in N$ consists of multiple physical machines (PMs) $n_z \in n$ with specific resource capacities $R(n_z)$. A VNF replica v_{kj}^i comes with certain resource demands $D(v_{kj}^i)$ for the underlying PM on which it is executed. This allows the network operator to implement vertical scaling for the different VMs, such that different VNF instances of the same VNF type may have different resource demands, according to their specific workload. Furthermore, two decision variables are introduced:

(1) $x_{v_{kj}^i}^{n_z}$: set to 1 if replica v_{kj}^i of VNF v_k^i is located at PM n_z of server $n \in N$
(2) $y_{uv}^{v_{kj}^i v_{k'j'}^i}$: the amount of traffic that flows between VNFs v_{kj}^i and $v_{k'j'}^i$ that is routed over link $(u, v) \in E$

In addition to that, $bw(u, v)$ describes the bandwidth capacities of link $(u, v) \in E$. Further notation is introduced to tailor to an IoT setting. The binary variable $\tau_{v_k^i} = \{0, 1\}$ indicates whether VNF v_k^i is an IoT-based VNF. This value is the same for every VNF instance v_{kj}^i. V_g is defined as the set of VNFs that are associated with a gateway $g \in G$, this includes exactly those VNFs that are IoT-based and requires for g to be connected to VNF's corresponding sensors.

It should be mentioned that most of the notation and the general infrastructure model is based on [7].

In this paper, dynamic network features are not considered in the system model itself, but will be addressed in the algorithms in later sections. For more information on how to integrate dynamic features into the problem formulation, please refer to related work such as [12] and [14].

4 PROBLEM FORMULATION

The objective function in 4.4 optimizes for three main objectives: 4.1 Availability of services, 4.2 Latency of a request and 4.3 System Cost of the current deployment.

4.1 Availability

The following availability model is based on the one presented in [13], but extends the proposed model to also respect VNF replicas. To describe the issue of availability, further notation is introduced. Let $q_{v_{kj}^i}$ be the probability that instance v_{kj}^i of VNF v_k^i fails, q_n and q_{n_z} are the probabilities that node $n \in N$ and a PM n_z at node $n \in N$ fails, respectively. These probabilities are known out-of-band by the system operator through statistical models or prior experience. Because VNFs are executed inside VMs, a VNF (replica) fails, if either the PM on which it is run fails or the VNF instance itself goes down. At the same time, a PM n_z on a physical node $n \in N$ becomes unavailable, if the entire data center n fails. The probability that VNF instance v_{kj}^i is available can be described as follows:

$$a_{v_{kj}^i} = \sum_{n \in N} \sum_{n_z \in n} x_{v_{kj}^i}^{n_z} (1 - q_n)(1 - q_{n_z})(1 - q_{v_{kj}^i}) \tag{1}$$

Equation 1 is available if neither the PM n_z at node $n \in N$ fails, nor does the VM on which VNF v_{kj}^i is executed itself fail. The binary variable $x_{v_{kj}^i}^{n_z}$ should only be set to 1 for exactly one PM n_z of any node n and thereby ensures that the probability for the right PM and VNF replica are associated with each other.

VNF v_k^i is available if at least one of its instances v_{kj}^i is available:

$$\begin{aligned} a_{v_k^i} &= 1 - Pr\{\text{All replicas of VNF } v_k^i \text{ fail}\} \\ &= 1 - \prod_{v_{kj}^i \in v_k^i} (1 - a_{v_{kj}^i}) \end{aligned} \tag{2}$$

The availability of an entire SC S^i can be described as the probability that all constituting VNFs v_k^i of SC S^i are available:

$$a_{S^i} = \prod_{v_k^i \in S^i} a_{v_k^i} \tag{3}$$

To describe the availability of the entire system, none of the SCs S should fail:

$$A(x) = 1 - \prod_{S^i \in S} (1 - a_{S^i}) \tag{4}$$

4.2 Latency

The latency model will combine models from [7] and [10]. The general delay distinction is based on [7] and [10] is utilized to respect IoT settings in the latency model. To model the network behaviour, [7] characterizes four different delay sources:

(1) Propagation delay $d_p(u,v)$ describes the link delay of edge $(u,v) \in E$ for one bandwidth unit.
(2) Transmission delay $d_{tr}(u,v)$ can also be considered as the aggregation time of device $u \in I$. This is generally understood as the time one bandwidth unit needs to be transferred from communicating device u to its outgoing link (u,v).
(3) The processing delay $d_{pr}(v^i_{kj})$ is the time needed by VM j hosting the VNF v^i_{kj} to apply a specific network feature on arriving packets.
(4) Queuing Delay $d_q(n_z)$ is the time packets spend in a PM n_z of a node $n \in N$ of the network while traversing the SC path.

Paper [7] further identifies an unavoidable latency for management tasks like creating, booting and loading of VNFs.
The latencies of queuing and processing delay for a SC S^i can be described as:

$$L_{S^i,P}(x) = \sum_{v^i_k \in S^i} \sum_{v^i_{kj} \in v^i_k} \sum_{n \in N} \sum_{n_z \in n} x^{n_z}_{v^i_{kj}} \cdot (d_{pr}(v^i_{kj}) + d_q(n_z)) \quad (5)$$

Equation 5 iterates over each VNF instance v^i_{kj} of SC S^i and sums for every node $n \in N$ the delay spent inside of PM n_z. Variable $x^{n_z}_{v^i_{kj}}$ correlates the VNF replicas v^i_{kj} with the corresponding PMs n_z of node $n \in N$.
The latencies for transmission and propagation delay is defined as follows:

$$L_{S^i,uv}(y) = \sum_{v^i_k \in S^i} \sum_{k'>k, v^i_{k'} \in S^i} \sum_{v^i_{kj} \in v^i_k} \sum_{v^i_{k'j'} \in v^i_{k'}} \sum_{(u,v) \in E} y^{v^i_{kj} v^i_{k'j'}}_{uv} \cdot (d_p(u,v) + d_{tr}(u,v)) \quad (6)$$

This equation considers all the edges located in the network in a predefined order $k' > k$. With $y^{v^i_{kj} v^i_{k'j'}}_{uv}$ the latency $(d_p(u,v)+d_{tr}(u,v))$ is scaled to the amount of traffic flowing between two VNFs $v^i_{kj} v^i_{k'j'}$.
As already noted, IoT-based VNFs may send control messages to its associated sensor, thus the latency model should also consider the delay of control messages from IoT-based VNFs back to the sensor.

$$L_{S^i,IoT}(x) = \sum_{v^i_k \in S^i} \sum_{v^i_{kj} \in v^i_k} \sum_{(g,v) \in E'} \tau_{v^i_k} \cdot x^v_{v^i_{kj}} \cdot (d_{tr}(g,v) + \Delta_g) \quad (7)$$

E' is defined as $E' = \{(u,v) \in E | u \in G, v \in N\}$, being all the edges from gateways $g \in G$ to nodes $v \in N$. Furthermore, the aggregation time for a specific gateway $g \in G$ is given by Δ_g and is known by the network operator. The equation can be understood as the latency of all the control messages of IoT-based VNFs v^i_k of a specific SC S^i that are associated with sensors of gateways $g \in G$. The delay of one such message is given by the transmission delay from the node that hosts the VNF to the gateway g and the aggregation time Δ_g of gateway g. The latency between the gateway and the corresponding sensor is negligible and thus not included in this model. In general, the management of the sensor network of a specific gateway is not conducted by the network operator.
In the end, the overall latency of the system is provided by:

$$L(x,y) = \sum_{S^i \in S} L_{S^i,P}(x) + L_{S^i,uv}(y) + L_{S^i,IoT}(x) \quad (8)$$

The latency can be described as the summation of latencies for all service chains $S^i \in S$. Equation 8 optimizes for both decision variables x and y. For each SC S^i, the equation takes into account the time $L_{S^i,P}(x)$ spent in specific nodes along the path belonging to S^i, the latency $L_{S^i,uv}(y)$ for transmitting corresponding messages along edges $(u,v) \in E$ of SC path S^i and the delay $L_{S^i,IoT}(x)$ of all control messages for IoT sensors of a SC S^i.

4.3 System Cost

The cost model adapts the model from [10]. Further notation is needed to describe costs of system operation:

(1) $c^{com}_{n_z}$: The cost of one computing resource unit at PM n_z of node $n \in N$.
(2) c^{net}_{uv}: The cost of one bandwidth unit over link $(u,v) \in E$.

The resource demand for a specific PM n_z of node $n \in N$ is given by:

$$R_{n_z}(x) = \sum_{S^i \in S} \sum_{v^i_k \in S^i} \sum_{v^i_{kj} \in v^i_k} x^{n_z}_{v^i_{kj}} \cdot D(v^i_{kj}) \quad (9)$$

The equation simply adds up all the resource demands $D(v^i_{kj})$ of those VNF replicas v^i_{kj} that are situated at PM n_z of node $n \in N$.
The bandwidth demand of edge $(u,v) \in E$ is given by:

$$B_{uv}(y) = \sum_{S^i \in S} \sum_{v^i_k \in S^i} \sum_{k'>k, v^i_{k'} \in S^i} \sum_{v^i_{kj} \in v^i_k} \sum_{v^i_{k'j'} \in v^i_{k'}} (y^{v^i_{kj} v^i_{k'j'}}_{uv} + y^{v^i_{kj} v^i_{k'j'}}_{vu}) \quad (10)$$

For a specific edge $\{u,v\} \in E$, the consumed bandwidth of all SCs $S^i \in S$ is computed. The equation will iterate over all SCs S^i and add the bandwidth consumed between each two communicating VNFs $v^i_{kj}, v^i_{k'j'}$. Because the network is formulated as a directed graph, both edges $(u,v), (v,u) \in E$ must be considered.
The bandwidth demand of sensor information of a gateway g and its IoT-based VNFs at a specific node $n \in N$ is defined as:

$$B_{gn}(x) = \sum_{n_z \in n} \sum_{v \in V^{n_z}_g} \sum_{v_j \in v} x^{n_z}_{v_j} \cdot b^{sen}_{v_j} \quad (11)$$

$V^{n_z}_g$ is all the IoT-based VNFs at PM n_z at node $n \in N$ correlated with gateway $g \in G$. Thus, this construction can omit the additional IoT check with τ. Let $b^{sen}_{v_j}$ be the network bandwidth between instance v_j of VNF $v \in V^{n_z}_g$ and sensors associated with g, this includes data sent by gateway g and control messages arriving at g. Equation 11 calculates the sensor bandwidth demand of a specific gateway $g \in G$ to a node $n \in N$.
The overall system cost can be constructed as follows:

$$C(x,y) = \sum_{(u,v) \in E} B_{uv}(y) \cdot c^{net}_{uv} + \sum_{(g,n) \in E'} B_{gn}(x) \cdot c^{net}_{gn} + \sum_{n \in N} \sum_{n_z \in n} R_{n_z}(x) \cdot c^{com}_{n_z} \quad (12)$$

In equation 12, the resource and bandwidth demands are scaled with the respective costs $c^{net}_{uv}, c^{com}_{n_z}$ to calculate the overall deployment cost. The cost function $C(x,y)$ optimizes for both variables x and y.

4.4 Objective Function

There are basically two different approaches on how to construct the objective function:

(1) From a network operator's standpoint, thus minimizing system cost or
(2) From a customer's perspective, thereby optimizing for latency and availability.

In this survey, I decided to take the customer's perspective. Not only is latency itself rather an optimization goal than a constraint, but the paper should also focus more on the service quality assurance of IoT services than minimizing the operation cost for network operators. Thus, the objective in this survey is to optimize for latency and availability similar to [13]. At the same time, system cost should be constrained to the operator's need. Furthermore, concepts of [7] and [10] are also integrated into the following objective function:

$$minimize - \{x, y\} : w_l \cdot L(x, y) - w_a \cdot A(x) + \sum_{S^i \in S} \sum_{v_k^i \in S^i} \sum_{v_{kj}^i \in v_k^i} \sum_{n \in N} \sum_{n_z \in n} w_{n_z} \cdot x_{v_{kj}^i}^{n_z}$$

$$+ \sum_{S^i \in S} \sum_{v_k^i \in S^i} \sum_{k' > k, v_{k'}^i \in S^i} \sum_{v_{kj}^i \in v_k^i} \sum_{v_{k'j'}^i \in v_{k'}^i} \sum_{(u,v) \in E} w_{uv} \cdot y_{uv}^{v_{kj}^i v_{k'j'}^i} \quad (13)$$

$$subject\ to : R_{n_z}(x) \leq R(n_z), \forall n_z \in n, \forall n \in N \quad (14)$$

$$B_{uv}(y) \leq bw(u, v), \forall (u, v) \in E \quad (15)$$

$$B_{gn}(x) \leq bw(g, n), \forall (g, n) \in E' \quad (16)$$

$$C(x, y) < E \quad (17)$$

$$\prod_{v_k^i \in S^i} a_{v_k^i} > A_{IoT} \cdot \mathbb{1}(\sum_{v_k^i \in S^i} \tau_{v_k^i} > 0), \forall S^i \in S \quad (18)$$

$$A(x) > A \quad (19)$$

$$\sum_{n \in N} \sum_{n_z \in n} x_{v_{kj}^i}^{n_z} = 1, \forall v_{kj}^i \in v_{k'}^i, \forall v_k^i \in S^i, \forall S^i \in S \quad (20)$$

$$\sum_{q \in Q} x_{v_{kj}^i}^{q} = 0, \forall v_{kj}^i \in v_k^i, \forall v_k^i \in S^i, \forall S^i \in S \quad (21)$$

$$\sum_{n \notin T} x_{v_{kj}^i}^{n} = 0, \forall v_{kj}^i \in v_k^i, \forall v_k^i \in S^i, \forall S^i \in S \quad (22)$$

$$x_{v_{kj}^i}^{n_z} \in \{0, 1\}, \forall n_z \in n, \forall n \in N, \forall v_{kj}^i \in v_k^i, \forall v_k^i \in S^i, \forall S^i \in S \quad (23)$$

The objective function 13 itself optimizes for opposing goals of latency and availability. The system operator can implement a policy $w_l + w_a = 1$ to determine the relative importance of latency vs availability. This allows approaching uRLLC requirements of low latency and high service availability if both are set to $w_l = w_a = 0.5$[13]. The last two terms of the objective function utilize the weights w_{n_z} and w_{uv} to enable load balancing. The weight $w_{nz} = \frac{1}{R(n_z)}$ is set to the inverse of the available capacity of PM n_z, $w_{uv} = \frac{1}{bw(u,v)}$ is set to the inverse capacity of link $(u, v) \in E$. This construction allows for less loaded resources to be selected, according to [7].

Following the objective function, multiple constraints are defined. Constraint 14 ensures that MEC and cloud resources are not oversubscribed. The resource demand $R_{n_z}(x)$ for a specific PM n_z of a network node $n \in N$ should not exceed the resource capacities $R(n_z)$ of that PM. Similarly, 15 and 16 ensure that link resources are not oversubscribed. This also respects control messages of IoT-based VNFs to their associated sensor over gateway g in 16. 17 represents the budget constraint, this limits the cost of service deployment to below a pre-determined cost E. In 18, a minimum availability rate A_{IoT} must be satisfied by any IoT-based service chain, i.e. at least one VNF v_k^i in SC S^i must be IoT-based. Furthermore, a general availability minimum A for the entire system must be satisfied by a final deployment in 19. Budget limit E and availability minimum A are pre-defined by the network operator and may be based on SLAs between the network operator and its tenants. The last couple of constraints provide additional constraints for proper network operation. Constraint 20 enforces that a particular instance v_{kj}^i of VNF v_k^i is only deployed at exactly one network node $n \in N$. Furthermore, 21 compels these VMs to only be deployed on computing nodes $n \in N$ and not on routers $q \in Q$. Constraint 22 enforces the final solution to only allow deployments that are not in violation to the location-constraints of specific VNFs T. The last constraint 23 ensures that the decision variable $x_{v_{kj}^i}^{n_z}$ is constrained to binary variables. At the same time $y_{uv}^{v_{kj}^i v_{k'j'}^i}$ is not confined by any constraint and can adopt any value. This renders the optimization problem as a Mixed Integer Linear Program (MILP), which can be solved by powerful mathematical solvers, e.g. CPLEX.

As the system model presented in this paper attempts to model the infrastructure in as much detail as possible, the search space for the optimal solution becomes quite large. Not only is the general VNF placement and chaining problem known to be NP-hard, but the integration of IoT features adds additional complexity to the problem. Although optimization problems can be solved by CPLEX implementations, the time complexity to obtain these optimal solutions is not suitable for real-world implementations, especially for a large search space. It is therefore much preferred to generate sub-optimal solutions that only approximate the optimal one, but achieve good performance and scalability. In the following section, several algorithms are presented that reduce the complexity of the objective function 13 and associated constraints 14-23.

Before introducing different algorithms and heuristics, it should be mentioned that some of the complexity of the objective function and its constraints can be avoided by abstracting away specific details of the network. Instead of considering multiple instances v_{kj}^i for one VNF v_k^i, it might be reasonable to only allow a VNF v_k^i to be deployed at exactly one physical node. This obviously rids the system model of the ability to express horizontal scaling of VMs. It might also seem natural to ignore the different physical machines n_z of one computing node $n \in N$ in the network. The reasoning behind such a decision would be that the management and operation of these different PMs on a physical node might not fall into the direct responsibility of the network operator itself, therefore can be ignored by the system model. These two optimizations on their own cancel out a summation symbol each in most of the computing intensive terms of the objective function and its associated constraints. Further simplifications can be undertaken for the different types of network nodes. Although a network consists of computing

nodes N (i.e. servers) and forwarding nodes Q (i.e. routers), the latter might be ignored in the system model due to performance reasons. This boost in performance obviously comes with a loss in accuracy of the system model and the generated solutions from the objective function. The network operator should weigh the relative importance of modeling accuracy vs performance. Although it is possible to simplify even further for performance gains, it is not advised to do so. As the system model should fulfil a certain amount of accuracy to have noticeable effects in real-world scenarios, most of the expressed features in objective 13 and its constraints 14 - 23 should in fact stay untouched. As mentioned earlier, the system model should be able to express the different layers (IoT layer, MEC layer and Cloud layer), as well as explicitly integrate IoT characteristics into the model. The three key properties latency, availability and system cost should also stay intact as all of these features are of interest for actual deployment scenarios. And even if those details would be removed from the system model, this might still not be sufficient. Several papers from section 2 present a less detailed system model and they still had to provide better performing algorithms and heuristics, as the underlying problem is NP-hard on its own.

5 ALGORITHMS & HEURISTICS

In this section, different algorithm schemes are introduced and the overall approach is presented on how to derive a specific algorithm from the objective function. In 5.1, a specific meta-heuristic, called a Genetic Algorithm (GA), is introduced, adapted to the VNF placement and chaining problem and extended for the IoT in MEC. Meta-heuristics are considered to provide near-optimal solutions with acceptable performance bounds. They are guided random search techniques that combine a heuristics-based approach with randomization techniques. It finds near-optimal solutions by exploring the search space and intelligently uses past search experience to guide further search behaviour.[2] In 5.2, an algorithm that implements dynamic network behaviour is introduced and 5.3 presents algorithm schemes that respect VNF scaling during SC allocation. It should be noted that this paper attempts to bring across the general idea and approach behind constructing such algorithms for the VNF placement and chaining problem in MEC, rather than demonstrating a precise implementation-ready algorithm formulation. For accurate implementations, please refer to related work featured in section 2 or combine presented ideas from this survey with pseudocode examples from sources like *Wikipedia* or [2].

5.1 Genetic Algorithm (GA)

Genetic Algorithms fall into the class of Evolutional Algorithms[2]. They are inspired by the laws of natural selection from biology and can be utilized to solve optimization problems through simulation of genetic processes. Similar to biology, *crossover* and *mutation* are applied to a population of individuals over a certain number of generations. In this case, individuals are represented by solutions to the optimization problem. Solutions are encoded as chromosomes, which are sequences of genes. Each solution has a fitness value that shows the quality of the solution. A high fitness value increases the chances for the individual to be selected for reproduction to produce offspring with higher anticipated fitness values. Further simulation conditions are that the population is limited to a fixed size. To make room for new offspring, individuals with low fitness values are eventually replaced by newly generated solutions.

5.1.1 Why GA?. As can be seen in [2], there exists an endless list of meta-heuristics that can be applied to any type of optimization problem, thus can also be constructed for the VNF placement and chaining problem. In this seminar work, a GA was chosen for this problem. There are several reasons for that: The GA has gained popularity in research as well as industry because of its intuitiveness, ease of implementation and the ability to effectively solve highly nonlinear, mixed integer optimization problems.[6] As the VNF placement and chaining problem is an integer optimization problem, it seems natural to apply a GA to this problem. This assumption is supported by the fact that part of the objective function is based on [13], which also applies a GA scheme. The VNF placement and chaining problem can be understood as an optimization problem that operates on discrete values. As such, specifically three different meta-heuristics come to mind: Tabu Search (TS) Algorithm, already applied in [7], Simulated Annealing (SA) and Genetic Algorithm (GA).[2] The first two algorithms are single-solution based, while the GA is population-based. Generally, any of these algorithms can be applied for the VNF placement and chaining problem, but the GA provides the most intuitive way of adapting to this optimization problem.
The drawback of the GA is a rather bad computational cost. Especially during the phases of the GA, a lot of generations are wasted without much impact on the overall fitness of the population as the algorithm slowly converges towards the global optima. In this paper, a local search algorithm is utilized to provide an initial population of individuals with relatively high overall fitness. Thus, the GA begins operation from a better starting point and may converge faster.

5.1.2 Data Structure. A data structure needs to be provided on which the GA can operate. This data structure must integrate the VNF placement and chaining problem. This seminar work assumes the following basic data structure:

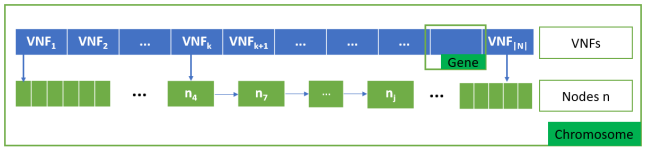

Figure 2: Data structure representing a solution (chromosome) for the VNF placement and chaining problem.

Each chromosome represents one valid solution to the optimization problem when applying the constraints 14 - 23. A data structure for such a solution (chromosome) is given in figure 2. Each chromosome is a sequence of genes, where each gene represents one VNF allocation in the network. As horizontal scaling should be respected, a specific VNF can be deployed on several physical nodes. The consolidation of a specific VNF_i and physical node n_j might represent a specific VNF instance. This is only the case if horizontal scaling would require these different VNF instances to be deployed on different physical nodes. Otherwise, the data structure from figure 2 would have to be extended accordingly.

5.1.3 Initial Population. As described before, the GA might have a bad computational complexity. The reason for that is the fact that the basic GA places individuals all over the search space and then slowly converges towards the global optima. An initial pool of solutions is created by generating random assignments of VNFs to physical nodes. All of these solutions should fulfil the constraints of the objective function 13. With this initial assignment, the GA will spend a lot of time on suboptimal solutions. Even a very basic local search algorithm could improve the performance of the GA. If such a local optimization would be applied to every individual the GA generates, the GA would only search the space of local optima. With this simple technique, the GA will greatly improve the chances of finding the best solutions to the optimization problem.

As with meta-heuristics, there are plenty of options to choose from for local search algorithms. For sake of simplicity, a basic hill climbing heuristic is utilized for this local optimization process[2]. This heuristic is an iterative algorithm that starts with an arbitrary solution and makes incremental changes to that solution. Every iteration of the algorithm, neighbour states are collected and their score is calculated by means of the fitness function, which will be elaborated on later. Neighbour states in this case can be understood as the movement of a specific VNF from one node to another node, thus representing one such incremental change to the current solution. The heuristic will loop for as long as such changes produce a better solution and will stop eventually if there is no neighbour state that exhibits higher fitness. The hill climbing heuristic is applied to every individual of the initial population. The resulting individuals will most likely not be optimal solutions but rather describe local optima. Nevertheless, these solutions provide a good starting point for the GA and should lead to a faster convergence on the part of the GA.

5.1.4 The Algorithm. The GA consists of the following phases[8]:

(1) **Initial Solution:** Create an initial pool of solutions with the techniques described in 5.1.3.
(2) **Fitness function:** To point the evolutional process into the right direction, a fitness function needs to be provided. The fitness function must be based on the objective function. In [13] a point (l, a) is applied to every gene, where l is the latency value and a is the availability value. The fitness value is defined as the euclidean distance between point (l, a) and ideal point (u_l, u_a). The ideal point represents an utopian case where latency value is $u_l \approx 0$ and availability value is $u_a \approx 100\%$. Although [13] applies the fitness function to every gene, in this paper this function is only applied to the entire chromosome. Not only should this design decision improve the performance of the overall algorithm, but the underlying crossover scheme between this paper and [13] is different. It might also be of interest to include system cost into the fitness function. In that case, a fitness value is represented as a point (l, a, c), where c stands for the cost of the considered solution. The fitness function thus would result in $\varepsilon = \sqrt{(u_l - l)^2 + (u_a - a)^2 + (u_c - c)^2}$. A small value for ε indicates that the solution exhibits a high fitness value as it is close to the optimal solution.
(3) **Selection:** In this phase, multiple pairs of individuals (parents) are selected based on their fitness value. Usually a crossover rate r_c determines how many individuals of the population are selected for reproduction on average. The individuals of the population are ordered according to their fitness value and the ones with the best fitness value are selected to produce offspring. This means that individuals with a higher fitness value have a higher chance of being selected for reproduction. Selected parents will pass their genes on to the next generation.
(4) **Crossover:** Crossover generates new offspring and potentially creates offspring with higher fitness value. Each offspring will feature genes of both of its parents. For each of the parents that was selected to reproduce, a random crossover point inside their genes is chosen. A pair of parents produces offspring by exchanging their genes with each other up to the respective crossover point. The genes following the crossover point are blindly copied into the respective offspring. This results in offspring featuring genes of both of their parents. For the VNF placement and chaining problem, a gene will only be exchanged if the resource constraint of a node is satisfied. This crossover process is demonstrated in figure 3.

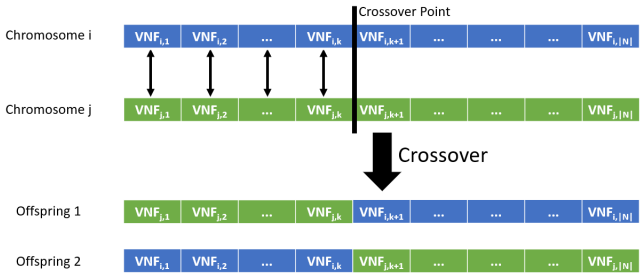

Figure 3: Crossover

(5) **Mutation:** There is also a low probability that one of the offspring mutates. This probability is given as the mutation rate r_m. A mutation for a chromosome means that the allocation schemes of one VNF are exchanged with the one of another VNF, if and only if the constraints are still satisfied after the swap. Mutation is necessary to maintain diversity within the population and prevent premature convergence to a sub-optimal solution. The mutation process is visualized in figure 4.

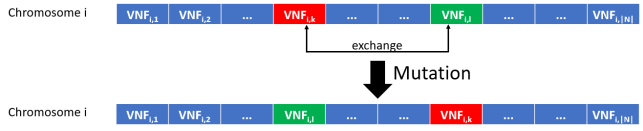

Figure 4: Mutation

The GA either runs for G generations or will terminate if the population has converged, such that offspring does not feature much better fitness than the rest of the population.

5.1.5 Complexity/Scalability. Let P be the population for the GA and S the VNFs to allocate on N nodes in the network.

(1) **Initial Solution:** First, random assignments are generated for each individual of the population P. Therefore, t_a is defined as the constant time needed to generate a random assignment of a specific VNF $v \in S$ to nodes $n \in N$. This results in $O(|P| \cdot |S| \cdot t_a) = O(|P| \cdot |S|)$.
The time complexity analysis must also account for the Hill Climbing heuristic. This heuristic is applied to every individual of P. The hill climbing heuristic can be interrupted at any time and returns the best result encountered up to this point. It takes t_h constant time to apply the heuristic to a specific individual. Thus, this results in $O(|P| \cdot t_h) = O(|P|)$ for the hill climbing heuristic.
As a result, $O(|P| \cdot |S|) + O(|P|) = O(|P| \cdot |S|)$ time is spent on generating an initial solution.
(2) **Fitness Function:** Let t_f be the time spent to calculate the fitness value for a specific individual. The fitness function is applied to every individual of P on every generation. This results in $O(|P| \cdot t_f) = O(|P|)$.
(3) **Selection:** All individuals of population P must be ordered according to their computed fitness value. As such, a standard sorting algorithm needs to be applied that leads to $O(|P| \cdot log|P|)$.
(4) **Crossover:** The selection phase decides on $r_c \cdot |P| = O(|P|)$ individuals that are selected for reproduction. Furthermore, t_c is assumed as the time spent on performing the exchange operation for two respective VNFs on two selected parents. This exchange operation will be applied $O(|S|)$ times depending on the crossover point. As a result, the crossover phase will take $O(|P| \cdot |S| \cdot t_c) = O(|P| \cdot |S|)$ time.
(5) **Mutation:** As only a small probability r_m is associated for mutation of an offspring, the time spent on mutation is negligible, thus $O(c) = O(1)$.

The GA will run for G generations. The time complexity for the entire GA is defined as:

$$\begin{aligned} T &= O(|P| \cdot |S|) \text{ // } \textit{initial solution} \\ &+ G \cdot (O(|P|) \text{ // } \textit{fitness function} \\ &+ O(|P| \cdot log|P|) \text{ // } \textit{selection} \\ &+ O(|P| \cdot |S|) \text{ // } \textit{crossover} \\ &+ O(1)) \text{ // } \textit{mutation} \\ &= O(|P| \cdot |S| + G \cdot (|P| \cdot log|P| + |P| \cdot |S|)) \end{aligned} \tag{24}$$

It should be mentioned that the analysis in 24 is only a rough approximation. The GA would need to be evaluated in a real-world implementation to understand the actual time complexity.
The computational complexity of a different GA scheme was already demonstrated in [13]. They further evaluate their GA version for different node types, e.g. edge or central cloud, and demonstrate the positive impact of edge nodes on deployment delays. The GA scheme was compared with a CPLEX implementation. It is shown that the GA exhibits good scalability as the number of nodes increase. Furthermore, the GA represents a good approximation to the optimal solutions of the CPLEX implementation while taking noticeably less time to generate potential allocation schemes.[13]

5.1.6 Integration of the IoT. As the GA makes use of the objective function 13 and constraints 14-23, basic IoT features are already integrated through the system model. In order to explicitly respect IoT features in the Genetic Algorithm, it is necessary to extend the former algorithm scheme and the associated data structure. As described in earlier sections, certain VNF types can be associated with IoT services. These IoT-related VNFs should preferably be placed at MEC nodes $n \in M$. This information is known by the network operator in advance and can be integrated into the allocation scheme by policy. For the GA, this translates to the restriction of IoT-related VNFs to being only deployed at MEC nodes. The according placement restrictions must be respected during the initial phase of the GA, as well as for the crossover and the mutation phase. If a VNF type belongs to a computing intensive task, these VNFs should rather be assigned to the data centers in the core cloud. This means that for each VNF, it might be useful to define a list of preferred nodes to be assigned to. If the resource constraints of a VNF for each of the respective nodes in this list can not be fulfilled, then the algorithm can still fall back to other less optimal nodes.

5.1.7 Alternatives. As already mentioned, the GA might have a bad computational complexity, thus would scale badly for larger problems. It would be of interest to see how single-solution based schemes like SA and TS algorithm would perform. Another option, not further considered in this paper, is the utilization of Particle Swarm Optimization (PSO), which is shown to have a better performance than the GA for most optimization problems[6]. As the underlying optimization problem itself represents a discrete search space, meta-heuristics like PSO, that operate on continuous search spaces, generally can not easily be applied to the VNF placement and chaining problem. Furthermore, they would probably provide lower quality solutions for integer constraint problems[6]. But there are ways to discretize the general PSO scheme[11]. Thus, PSO might still be a candidate for the VNF placement and chaining problem. GA and PSO are both population-based heuristics and both represent intuitive approaches to optimization problems.
PSO is based on the social foraging behaviour of some type of animals (e.g. birds, fish) and utilizes swarm intelligence to find a good approximation for the optimal solution. The different individuals of the swarm, also referred to as particles, move through the search space following the fitter members of the swarm. The search space is represented by a multi-dimensional hyper-volume as can be seen in figure 5.[2]

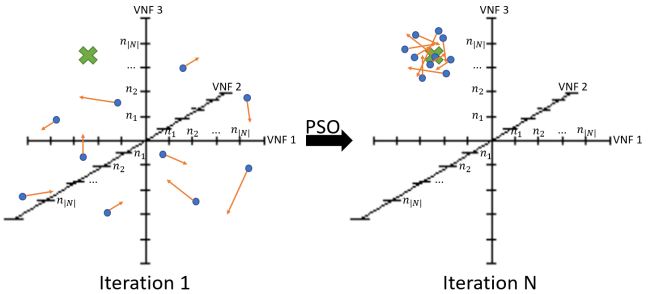

Figure 5: PSO

In order to apply the PSO to the VNF placement and chaining problem, it would be necessary to provide an underlying data structure. As the PSO operates in a multi-dimensional hyper-volume, a vector $S = [VNF_1, VNF_2, ...]^T$ is introduced that includes all VNFs for all SCs in the system. Each entry in S can be understood as its own dimension in the hyper-volume and is represented by a certain axis in the coordinate system. This means that each VNF_k can take an integer value in the range of $l \in \{1, 2, ..., |N|\}$ and as such references a specific node $n_l \in N$. This notation is based on the one presented in [13]. Each particle i in the swarm is represented by a location vector $p_i = S$ and a velocity vector v_i, which moves the particle to another location in the search space. Each particle moves around the search space based on the current position, the best position known to the particle as well as the global best solution of the entire swarm. Every move of a particle, the best solutions are updated according to the objective function. In the end, all the particles should converge and locate around the global optimal solution in the search space. The best solution known to the swarm will be the final solution to the optimization problem.

5.2 Algorithms For Dynamic Behaviour

It was shown in former sections, that respecting dynamic behaviour of networks in VNF placement algorithms can be crucial for the IoT. A first approximation for dynamic behaviour can be achieved by implementing a batch-like processing scheme [14]. This means that over a pre-defined time interval, all service requests are collected as batches and afterwards processed together by applying the optimization algorithm. Thereafter, the controller will deploy the VNF chains to the specified locations of the optimization result. [14] presented a well suited heuristic for such a dynamic batch-like scheme. They define a specific set R for IoT related service requests, as well as the set of possible destinations D an IoT device might move to. Furthermore, there exist probabilities ρ_{rd} that service request $r \in R$ visits destination $d \in D$. These probabilities are pre-calculated with statistics data and include precise mobility patterns of devices. The proposed heuristic will iterate over the set of collected requests $r \in R$ and find the destination $d \in D$ with highest probability ρ_{rd}, as well as the source node s at the edge that shows the lowest routing costs. Next, the shortest path between s and d is calculated and the VNFs of request r are allocated on the selected path. This heuristic will be applied to a specific batch after each collection period ends.[14] A formal model for such a time-slotted system is provided by [12].

5.3 Algorithms for VNF Scaling

As VNF scaling might be of interest for network operators, section 3 provided means of horizontal and vertical scaling in the system model itself. There is also the possibility to respect those scaling features within special algorithms as can be seen in [3].

5.3.1 Horizontal Scaling. Horizontal Scaling means that if multiple service requests generate service chains that share the same network function, then each SC will pass through a separate VNF instance of that function. For construction of the algorithm, it is defined that for each service chain, generated by a request, there exists a corresponding source node s and destination node d. To implement horizontal scaling, the algorithm will iterate over each service chain and find the shortest path from s to d utilizing a shortest path algorithm. The VNFs of the SC will be allocated along the selected path, respecting the resource constraints of the path's nodes. As this construction is repeated for every SC, multiple VNF instances are generated for one network function, if several service requests make use of it.

5.3.2 Vertical Scaling. In vertical scaling, VNF instances can serve multiple requests at the same time. This can be implemented by assigning more resources to a specific VM in which the VNF is executed. Similar to horizontal scaling, the algorithm iterates over the SCs in the system, finds a shortest path for that request and assigns the VNFs to specific nodes along the path. If a VNF is assigned to a node on which another VNF instance of the same type is already running, the algorithm simply assigns more resources to that VM, such that it can cope with the increased workload. Otherwise, a new VM for that VNF is generated on the respective node.

As this work does not provide an exact implementation of the presented algorithms, there will be no evaluation section in this paper. Nevertheless, it should be demonstrated how such an evaluation can be conducted. The derived algorithm needs to be compared with a reference algorithm. In the literature in section 2, they either compare their algorithm directly with the MILP implementation or they construct a randomized or greedy version of their algorithm and make a comparison with those algorithms. In some theoretical papers[5][12], they also proof the run-time of the algorithm. The evaluation is either conducted via simulation tools or a test bed is constructed to test the algorithm in a real-world implementation.

6 CONCLUSION

In this paper a rather detailed system model is provided for the VNF placement and chaining problem. The problem formulation differentiates the network into three different layers: the IoT layer, the MEC layer and the core cloud, thus integrating IoT features. The problem formulation yields the objective function 13 that is subject to constraints 14 to 23. To reduce the computational complexity of the constructed optimization problem, further algorithms and meta-heuristics were introduced that should provide a better computational bound for different design goals of the network. Specifically a GA scheme for the VNF placement and chaining problem was presented in detail. This GA version also integrated IoT features into the algorithm scheme itself. As was shown in the paper, it is a complicated endeavour to construct an efficient algorithmic scheme for the VNF placement and chaining problem. On the one hand, the model should provide enough detail to capture the nuances of a specific scenario, on the other hand the complexity quickly explodes in the light of the underlying NP-hard optimization problem. These opposing goals must be respected by the network operator in order to build a well-suited framework for real-world implementations. This survey should be seen as a holistic overview and guide over the process on how to construct a system model all the way to an algorithmic implementation for the VNF placement and chaining problem.

REFERENCES

[1] Solution Architect (Internet of Things / 5G Networking) Anurag Agarwal. 2017. *Optimizing Traffic for Emergency Vehicles using IOT and Mobile Edge Computing.* https://software.intel.com/content/www/us/en/develop/articles/optimizing-traffic-for-emergency-vehicles-using-iot-and-mobile-edge-computing.html

[2] Jason Brownlee. 2012. *Clever Algorithms: Nature-Inspired Programming Recipes.* https://raw.githubusercontent.com/clever-algorithms/CleverAlgorithms/master/release/clever_algorithms.pdf

[3] R. Gouareb, V. Friderikos, and A. Aghvami. 2018. Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization. *IEEE Journal on Selected Areas in Communications* 36, 10 (2018), 2346–2357.

[4] R. Gouareb, V. Friderikos, and A. H. Aghvami. 2018. Delay Sensitive Virtual Network Function Placement and Routing. In *2018 25th International Conference on Telecommunications (ICT).* 394–398.

[5] D. Harris, J. Naor, and D. Raz. 2018. Latency Aware Placement in Multi-access Edge Computing. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft).* 132–140.

[6] Rania Hassan, Babak Cohanim, and Olivier de Weck. 2005. A Comparison of Particle Swarm Optimization and the Genetic Algorithm, Vol. 2. https://doi.org/10.2514/6.2005-1897

[7] A. Leivadeas, M. Falkner, I. Lambadaris, M. Ibnkahla, and G. Kesidis. 2018. Balancing Delay and Cost in Virtual Network Function Placement and Chaining. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft).* 433–440.

[8] Vijini Mallawaarachchi. 2017. *Introduction to Genetic Algorithms — Including Example Code.* https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[9] Katta G. Murty. 2003. *Optimization Models For Decision Making: Volume 1 (Junior Level).* http://www-personal.umich.edu/~murty/books/opti_model/

[10] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet. 2020. Placement and Chaining for Run-Time IoT Service Deployment in Edge-Cloud. *IEEE Transactions on Network and Service Management* 17, 1 (2020), 459–472.

[11] John Sheppard Stephyn Butcher Shane Strasser, Rollie Goodman. 2016. A New Discrete Particle Swarm Optimization Algorithm. https://doi.org/10.1145/2908812.2908935

[12] Y. Xie, S. Wang, and Y. Dai. 2019. Provable Algorithm for Virtualised Network Function Chain Placement in Dynamic Environment. In *2019 IEEE Global Communications Conference (GLOBECOM).* 1–6.

[13] L. Yala, P. A. Frangoudis, and A. Ksentini. 2018. Latency and Availability Driven VNF Placement in a MEC-NFV Environment. In *2018 IEEE Global Communications Conference (GLOBECOM).* 1–7.

[14] G. Zheng, A. Tsiopoulos, and V. Friderikos. 2019. Dynamic VNF Chains Placement for Mobile IoT Applications. In *2019 IEEE Global Communications Conference (GLOBECOM).* 1–6.