On Reconfiguring 5G Network Slices

Matteo Pozza, Patrick K. Nicholson, Diego Lugones,
Ashwin Rao, Hannu Flinck, and Sasu Tarkoma

***Abstract*—The virtual resources of 5G networks are expected to scale and support migration to other locations within the substrate. In this context, a configuration for 5G network slices details the instantaneous mapping of the virtual resources across all slices on the substrate, and a feasible configuration satisfies the Service-Level Objectives (SLOs) without overloading the substrate. Reconfiguring a network from a given source configuration to the desired target configuration involves identifying an ordered sequence of feasible configurations from the source to the target. The proposed solutions for finding such a sequence are optimized for data centers and cannot be used as-is for reconfiguring 5G network slices.**

We present *Matryoshka*, our divide-and-conquer approach for finding a sequence of feasible configurations that can be used to reconfigure 5G network slices. Unlike previous approaches, *Matryoshka* also considers the bandwidth and latency constraints between the network functions of network slices. Evaluating *Matryoshka* required a dataset of pairs of source and target configurations. Because such a dataset is currently unavailable, we analyze proof of concept roll-outs, trends in standardization bodies, and research sources to compile an input dataset. On using *Matryoshka* on our dataset, we observe that it yields close-to-optimal reconfiguration sequences *10X* faster than existing approaches.

I. INTRODUCTION

5G network slicing redefines the telecommunications market by converging the best of 5G connectivity and Network Function Virtualization (NFV) to enable new business opportunities in diverse domains such as mobile services, manufacturing, consumer and enterprise applications. While 5G can support applications with exceptional requirements for latency, throughput, and dense connectivity [1], [2], NFV enables the softwarization of Network Functions (NFs) . To fully leverage the benefits of 5G slicing, there is also a trend towards distributing NFs in a way that real-time functionality is deployed close to the access, whereas less critical functions run in the core network – preferably collocated in cloud to reduce costs. In this emerging scenario, there is a need to evolve traditional network-centric management towards more agile service-driven processes. That is, network operators need new cost-effective mechanisms to *adapt* network slices to varying, customer-specific workloads. In this context, adaptation implies dynamically auto-scaling slices to provision capacity, and to migrate their NFs to achieve the expected service performance [3]. The key challenge for operators is to concurrently coordinate customer slices individually within a much finer timescale (*e.g.,* a few times per day [4]) compared to traditional resource aggregation and over-provisioning practices.

Telecommunication networks leverage network functions to operate on their traffic. For example, network functions collect statistics, modify packet headers, or drop packets matching patterns of malicious traffic. Network functions built following the principles of NFV are called Virtual Network Functions (VNFs), and each network slice consists of several interconnected VNFs. Each VNF can have a variable number of instances, and each VNF instance runs as a Virtual Machine (VM).[1] These virtual resources need to be placed on the substrate such that the substrate is not overloaded and the Service-Level Objectives (SLOs) of all the network slices are met. We refer to the mapping of the virtual resources across all slices on the substrate as the *configuration*. A configuration details the location of the virtual resources in the network, and it is feasible if the SLOs of each slice are satisfied without overloading the substrate. NFV enables network operators to change the configuration. This may be required for various reasons including a slice is added or removed, the demands of a slice change, or some nodes of the substrate require maintenance.

Reconfiguring network slices consists of the following three steps: i) finding a new configuration, ii) finding an ordered sequence of feasible configurations from the current configuration to the new configuration, and iii) performing the actual migration of VMs. To increase business value, all three steps must be performed as fast as possible, thus the time budget is very small [7]. The first and the third step have received significant research attention [8], [9], [10]. In contrast, the problem of finding an ordered sequence of VM migrations has been studied in the context of data centers, however the proposed solutions cannot be used as-is when reconfiguring 5G network slices. For instance, Dow *et al.* [11] propose a variant of A* search, however, their algorithm does not simultaneously consider the bandwidth and latency requirements of the services. In Figure 1, we observe that their solution finds migration sequences only for a small number of network slice reconfiguration scenarios even when it was given a large time budget. This highlights the need for an algorithm that i) takes into account 5G requirements, and ii) is able to provide solutions efficiently.

In this paper we introduce *Matryoshka*, a divide and conquer algorithm to find a migration sequence for reconfiguring 5G slices. Our approach is designed to work at scale with large 5G

Matteo Pozza, Ashwin Rao, and Sasu Tarkoma are with the University of Helsinki.

Patrick K. Nicholson, Diego Lugones, and Hannu Flinck are with Nokia Bell Labs.

This is the author version that has been accepted for publication on January 28^{th}, 2020, in IEEE Journal on Selected Areas in Communications (JSAC) - Series on Network Softwarization & Enablers.

[1] In this work we use the term virtual machine (VM) to refer to a virtualized instance of a network function. Network functions can be virtualized using traditional VMs, or light-weight virtualization techniques such as containers [5] and Unikernels [6].

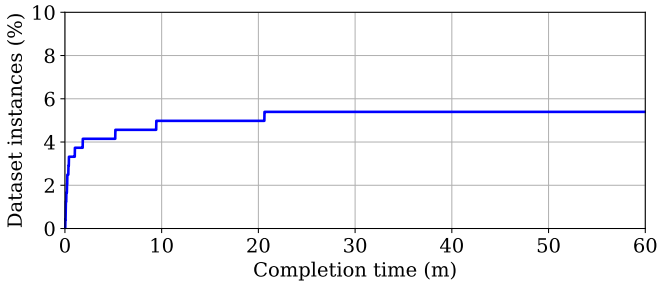


Fig. 1: **State-of-the-art algorithm: A* search.** *The algorithm used for finding a sequence of VM migrations to reconfigure virtual networks in data centers fails to scale with the stringent constraints of 5G networks. A low percentage (< 6%) of migration sequences is found even after running for more than one hour on high-end servers.*

networks by leveraging parallelism to reduce completion time. Initially, Matryoshka creates an abstraction of the configuration space in the form of a directed graph that we refer to as the *migration graph*, in which the edges represent VM migrations. Then, it runs a heuristic for traversing the migration graph over paths that do not violate the SLOs. When evaluating Matryoshka, we found that publicly available data about 5G deployments at scale is rather scarce. So, we collected information from specifications, telecommunications organizations and other sources to model the physical substrate, as well as the requirements of the network slices to generate sufficient reconfiguration examples to stress our algorithm. In summary, our contributions are:

- An algorithmic approach for creating appropriate migration strategies of virtual network functions running on 5G network slices. To the best of our knowledge, this is a first effort to understand *how* to change the configuration of such slices in a timely fashion while including critical 5G constraints in terms of bandwidth and latency.
- A modeling dataset of 5G slicing, for research purposes, which we use to represent potential substrates and create multiple network configurations for evaluating and comparing Matryoshka to state-of-the-art techniques.

Evaluating the performance of the approach of Dow *et al.* [11] for reconfiguring 5G network slices requires a dataset of pairs of source and target configurations. In §III, we detail the dataset we created for the network slices reconfiguration problem because such a dataset is currently unavailable for 5G networks. The dataset contains several problem instances, *i.e.,* examples of the network slices reconfiguration problem without solutions. Each problem instance is characterized by a pair of feasible source and target configurations; a description of a problem instance is presented in §III-D. We present and evaluate Matryoshka, our algorithm to efficiently solve the network slices reconfiguration problem in §IV and §V.

II. Background

In this section, we first provide an overview of network slicing and configuring slices in 5G networks. We then present an overview of the problem of reconfiguring 5G networks, and how A* search has been leveraged to address similar problems.

A. Network Slicing

5G networks will offer network services to a wide range of use cases including hand-held devices, self-driving cars, and e-health devices. These use cases are characterized by different requirements, *e.g.,* in terms of bandwidth and latency, and simultaneously satisfying all such requirements is a key challenge for network operators. The 3rd Generation Partnership Project (3GPP) has therefore proposed network slicing, *i.e.,* multiplexing logically independent networks on a substrate where each network is tailored for a single use case.

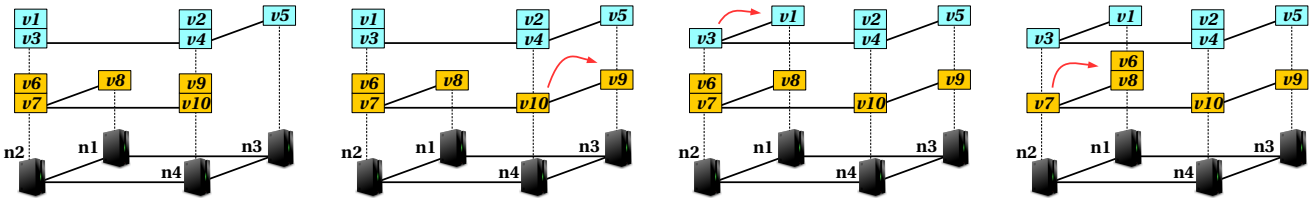
The virtual resources of each network slice can be divided into two groups. The first group comprises the virtual resources that handle basic services of the mobile network, such as registration and mobility. These virtual resources are typically shared between the network slices because their services are common across all use cases. For instance, a network operator can configure the Access and mobility Management Function (AMF) [12] to be shared across multiple slices. The second group comprises the virtual resources that are specifically designed to serve users of a specific use case, and consequently are not shared between network slices. These virtual resources are typically organized into chains of variable length [13]. In this work we focus on the second group, and we assume each use case is assigned an isolated virtual network which is instantiated on a shared substrate.

5G network slices are expected to use virtual network functions (VNF) which can be instantiated in any general-purpose computing node, thus allowing the slices to scale and adapt with the demands. A VNF is instantiated as Virtual Machines (VMs) or containers, and the number of VMs can be adjusted according to instantaneous network load. For example, an increase in the traffic load can trigger a firewall VNF to scale-out and instantiate additional VMs, while a decrease in the traffic load can trigger it to scale-in and decommission idle VMs.

B. Configuration for Network Slices

A configuration for network slices describes the mapping of the virtual resources across all slices on the shared substrate [14]. Specifically, it details the mapping of the VMs and virtual links across all slices on the computing nodes and links between the nodes. Figure 2a shows an example of two network slices configured on the same substrate. The first network slice depicted as blue VMs has VMs v1 and v3 mapped to host n2, VMs v2 and v4 mapped to host n4, and VM v5 mapped to host n3. The mappings of virtual links reflect the mappings of the VMs, *i.e.,* v3 and v4 have a virtual link which is mapped on the path between n2 and n4.

A given configuration can either be *feasible* or *infeasible*. A configuration is *feasible* if (i) the requirements of the corresponding use cases are met, and (ii) the physical network is not overloaded. Conversely, a configuration not fulfilling these two conditions incurs in violations of SLOs, and is therefore *infeasible*.



(a) *Source* (b) *After migration #1* (c) *After migration #2* (d) *Target reached*

Fig. 2: **Example of network slices reconfiguration.** *The substrate comprises hosts n1, n2, n3, n4 and their links, while the two network slices respectively comprise VMs v1, v2, v3, v4, v5 and links between them, and VMs v6, v7, v8, v9 and links between them. Network slices are initially configured as in Figure 2a and they must be reconfigured as in Figure 2d. Three migrations of VMs are required to obtain the target configuration. All configurations traversed during the reconfiguration must be feasible.*

C. Reconfiguring Network Slices

The scale-in and scale-out of the VNFs causes the configuration of network slices to change. For example, an aggressive scale-out policy can quickly saturate the capacity of a node which can result in SLO violations. To avoid such undesired configurations, network slices must be reconfigured to more desirable configurations.

The task of reconfiguring network slices involves the following three sub-tasks. The first sub-task is identifying a new configuration reflecting the objective of the mobile operator. For example, the mobile operator might be interested in distributing the load in the network as evenly as possible. The second sub-task consists in identifying a sequence of VM migrations to transform the network slice from the current configuration, called *source*, to the desired configuration, called *target*. Figure 2 shows an example in which reconfiguring the network slices involves three intermediate steps. In each step, exactly one VM is migrated from one node in the substrate to another. Note that it is essential to ensure that each intermediate configuration obtained is feasible. This ensures that (i) the reconfiguration does not violate any SLOs, and (ii) the network stays in a feasible configuration if the reconfiguration process is abruptly interrupted. The third and final sub-task is performing these VM migrations. In this paper we focus our attention on the second sub-task.

D. A* Search

A* search algorithm has been used by Dow *et al.* [11] to find a sequence of operations to reconfigure virtual networks in data centers. This algorithm guides the exploration for a path from a given source node to a target node with a heuristic that prioritizes nodes which are more likely to lead to the target node. The algorithm maintains two sets of nodes: an *openset* and a *closedset*. The openset contains all the nodes discovered by the algorithm whose neighbors haven't been discovered yet, while nodes whose neighbors have been discovered end up in the closedset. Each node in the openset has an associated priority. The algorithm starts from the source node and it repeats the following loop: (i) pick the node with the highest priority in the openset, (ii) discover its neighbors, (iii) assign a priority value to each neighbor, (iv) put each neighbor in the openset, and (v) put the node in the closedset. The priority value assigned to each node is a combination of its actual distance from the source node, which is computed while exploring the graph, and its distance from the target node, which is obtained through the heuristic.

III. Modeling Dataset

In this section we describe our best effort for modeling a large-scale 5G network. As existing data from previous generations of mobile technologies is not sufficient to understand 5G slicing end-to-end, we consulted multiple sources of information such as standardization entities and specifications, as well as 5G trials. Given the constant evolution of the standard, the modeling dataset may not necessarily reflect real-world parameters. However, the modeling methodology presented here can be used to incorporate updates and adapt to modifications.

In our attempt to model 5G slicing as thoroughly as possible, we first describe the expected features of the substrate and the capability of network slices. Then, we provide a definition of the *problem instance* for slice reconfiguration, and describe how we generate them to evaluate our proposal. Last, we discuss the validity of the modeling dataset in the context of our research.

A. Modeling Physical Substrate

1) Graph: In Figure 3, we present a high-level picture of a 5G network substrate. It exhibits a structure consisting of three levels: pre-aggregation, aggregation, and core. Nodes are organized in a ring in the first two levels, and in a mesh in the core level [15], [16], [17]. Nodes of different levels are interconnected in a hierarchical fashion. Each node in the aggregation ring is connected to at most seven nodes in the pre-aggregation ring, while each node in the core mesh is connected to at most two nodes in the aggregation ring [18]. The pre-aggregation ring corresponds to the so-called *edge* of the mobile network. The nodes of the pre-aggregation ring are called Centralized Units (CUs) and they correspond to the computing nodes to which base stations are directly attached [19]. We construct the network starting from the area to be covered. We obtain the number of CUs by dividing the whole area by the coverage area of a single CU, which we found to be 4 km^2 approximately.[2] We then calculate the

[2]The details on how the value was computed can be found in Appendix A.

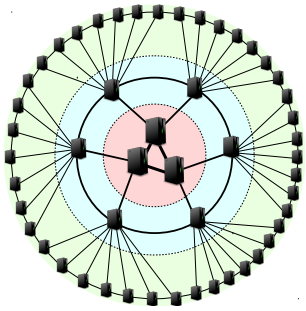


Fig. 3: **5G substrate.** *The network comprises three hierarchical levels: pre-aggregation ring (green), aggregation ring (cyan), and core mesh (red). The nodes in the pre-aggregation ring are the Centralized Units (CUs). In the figure, thicker links indicate higher available bandwidth. Similarly, bigger nodes indicate greater computational capacity.*

number of nodes in the aggregation and core levels, as well as the links between the nodes of each level.

In our study, we are interested in the part of the network where we can actually instantiate the VMs of the network slices. This feature can be offered by all nodes from the core to the pre-aggregation ring, the latter offering the lowest latency to the users [20], [21], [22]. Note that in 5G networks base stations are reduced to their minimal functionalities since the signal processing is offloaded to the CUs [23]. For this reason, they do not represent a good candidate for instantiating VMs of network slices and we do not represent them in our model.

2) CPU and memory: Each network node corresponds to a rack of servers, thus the number of CPUs and amount of memory available in the node is the sum of the CPUs and available memory of each server in the node. We have dimensioned each server with 28 cores and 64GB of memory, which is in line with the specifications of carrier-grade appliances [24]. The number of servers available at each node in the network depends on its hierarchy level, with resources at the end tending to be scarce [22]. Hence, we model nodes in the pre-aggregation ring with a small number of servers, *i.e.,* 4 servers, and increase this number towards the core, *i.e.,* x8 in the aggregation ring and x16 in the core mesh.

3) Bandwidth: Each network link represents multiple optic fiber connections, each of which has 100 Gbps bandwidth in both directions as predicted by commercial 5G forecasts [25], [26], [27], [28]. Similarly to the case of computational capacity, the total bandwidth available at each link decreases from the core to the pre-aggregation ring. We model the pre-aggregation ring, the aggregation ring, and the core mesh with 200 Gbps, 400 Gbps, and 800 Gbps links, respectively. Furthermore, the pre-aggregation ring and aggregation ring are interconnected through 200 Gbps links, while the aggregation ring and core mesh are interconnected through 400 Gbps links. Note that the bandwidth used in one direction is independent of the bandwidth used in the other direction of the link.

4) Latency: All links between nodes at the same hierarchy level incur a latency of 2 ms, whereas the latency between pre-aggregation ring and core mesh is approximately 10 milliseconds [27]. The physical distance between the aggregation ring nodes and core mesh nodes is approximately four times the distance between the pre-aggregation and aggregation ring nodes [20]. As a consequence, links interconnecting the pre-aggregation ring to the aggregation ring incur a latency of 2 ms, while links interconnecting the aggregation ring and core mesh have a latency of 8 ms. Note that each node adds a latency of approx. 50 μs when receiving, processing, and forwarding data packets [29], [30].

Use cases	**Bandwidth / km^2**		**E2E Latency**
	Download	Upload	
eMBB	750 Gbps	125 Gbps	10 ms
URLLC	10 Gbps	10 Gbps	1 ms
mMTC	100 Gbps	100 Gbps	50 ms

TABLE I: **Requirements for 5G use cases [1], [31].** *E2E latency corresponds to the one-way latency in the path from the user to the last VM of the network slice serving the user.*

B. Modeling Network slices

1) Graph: Figure 4 illustrates an example of a network slice. Each network slice is comprised of a number of VNFs linked or chained to each other. Each VNF can be instantiated over one or more VMs to scale according to the traffic load. The VMs of a VNF typically need not communicate with each other as they are expected to run independently. Instead, when considering two adjacent VNFs, each VM of the first VNF is linked to each VM of the second VNF. Thus, links between pairs of adjacent VNFs form complete bipartite graphs.

2) CPU and memory: The amount of CPUs and memory required by a network slice corresponds to the sum of CPU and memory requirements of its VNFs, while the requirements of a VNF add up to the amount of CPU and memory in all its VMs. Each VM is modeled to have 4 CPUs and 4 GB of memory [32]. We use the number of CPUs and the amount of memory to abstract fine-grained metrics that can be leveraged to identify if a node can host additional VMs. As an example, the CPU ready value recorded among the VMs running in a node indicates much more reliably if a node can host additional VMs or not [33]. The metrics we used do not affect the quality of our dataset because our objective is to obtain pairs of feasible configurations.

3) Bandwidth: Table I groups the estimated 5G requirements for three well-known *macro* use cases: i) enhanced mobile broadband (eMBB), ii) ultra-reliable low-latency communication (URLLC), and iii) massive machine-type communication (mMTC). These macro scenarios abstract out specific application use cases whose requirements would fall within a combination of these macro use cases. For example, smart buildings have requirements from both eMBB and mMTC use cases, while augmented reality belongs to both eMBB and URLLC use cases [34]. The bandwidth values in Table I are for each pair of adjacent VNFs in the corresponding network slices, and the total bandwidth is split evenly among all VM links when a VNF consists of multiple VMs. Figure 4 illustrates the bandwidth splitting. Assume the network slice serves mMTC users, and consider VNF_1 and VNF_2. Then, the bandwidth requirements for download, *e.g.,* from VNF_2 to

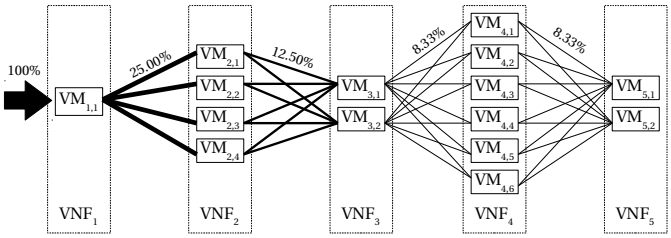


Fig. 4: **Example of network slice.** *Each slice comprises a sequence of VNFs, and each VNF is composed of a number of VMs. Given two adjacent VNFs, the links between their VMs form a complete bipartite graph, and the required bandwidth is split evenly between the links.*

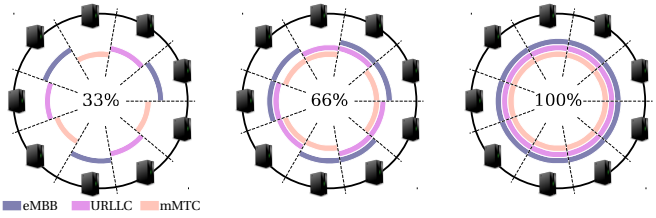


Fig. 5: **Density of use cases in pre-aggregation ring.** *Each CU serves either a single use case (left), two use cases (center), or all three use cases (right). The percentages indicate the fraction of CUs serving each use case. Note that a CU can simultaneously serve multiple use cases.*

VNF_1 is 100 Gbps, and the same holds for upload, *e.g.,* from VNF_1 to VNF_2. If VNF_1 is instantiated with a single VM and VNF_2 is instantiated with four VMs, there are four links between VNF_1 and VNF_2. As a consequence, the bandwidth splits evenly among the links, so that each link gets 25% of the overall amount, *i.e.,* 25 Gbps in both directions.

4) Latency: Table I also illustrates the latency requirements of all use cases. Latency requirements are end-to-end (E2E), thus implying that the total latency incurred in the path from the user to the last VNF of the network slice must yield below the indicated threshold. The same latency figures apply for the opposite path as well, *i.e.,* from the last VNF of the network slice to the end-user. Since we consider only the portion of the network behind the CUs, the latency between the UE and CU is subtracted from total E2E latency requirement. Such UE-to-CU latency amounts to 0.5 ms [20].

C. Scenarios and User distribution

The end users of each scenario can be distributed across the network in various ways. From an operator perspective, such a distribution is reflected by the CUs that are associated to each scenario. For each CU associated with a given use case, the operator would deploy a network slice for serving the users connected to that CU. The fraction of CUs associated with a certain use case is henceforth called *density*. Note that a CU can be associated to more than one use case simultaneously.

D. Problem instances

With the purpose of evaluating our algorithm and state-of-the-art techniques, we create multiple input scenarios to solve. We first define what a problem instance is and consists of. We then iterate over its components, providing for each of them the values that have been used.

1) Definition: A problem instance consists of two input configurations, along with the details of the substrate and the network slices. The two input configurations are a *source* configuration, *i.e.,* how VMs are placed in the network initially, and a *target* configuration, *i.e.,* how VMs should be placed at the end of the reconfiguration procedure. Solving the problem instance entails finding a sequence of VM migrations that allow reconfiguring the network slices from source to target without violating SLOs or overloading the network in the process. Next, we analyze the additional parameters we used for defining the substrate, the network slices, as well as source and target configurations of each problem instance.

2) Substrate: We defined four substrates by varying the geographical coverage area of the 5G network in 25, 50, 75, and 100 km^2. As discussed, the coverage area of a single CU is approximately 4 km^2. Thus, the four generated substrates have 7, 13, 19, and 25 CUs in the pre-aggregation ring, respectively.

3) Network slices: We consider different densities of use cases for the three scenarios depicted in Figure 5. In all these scenarios, there are two aspects to consider: i) the number of CUs serving a use case is always the same for the other use cases, and ii) a CU can serve multiple use cases simultaneously. In the first case, each CU serves only a single use case (either eMBB, URLLC, or mMTC), thus each use case has 33% of the CUs associated to it. In the second case, each CU serves two use cases, *e.g.,* eMBB and URLLC, thus each use case has 66% of the CUs associated to it. In the latter case, each CU serves all three use cases simultaneously, and therefore each use case has 100% of the CUs associated to it.

4) Source and target configurations: Given the substrate and a number of network slices from the modeling dataset, there is a high number of possible configurations. This is due to fact that each VM can be potentially placed in any node of the physical substrate. We therefore need some guidelines on how to select source and target configurations for the problem instance. Specifically, our aim is to represent a scenario in which network reconfiguration is required.

One such scenario is when a network operator is interested in reconfiguring the network to use spare resources at the edge. For this context, the source configuration represents a configuration where a share of the resources at the edge are unused. Conversely, the target configuration represents a configuration in which the resources at the edge are leveraged as much as possible. For the source, we identify the configuration whose CPU usage in the pre-aggregation ring is as low as possible. This way, CUs are likely to have spare resources which indicates a yet unexploited opportunity of consolidating the VMs at the edge. For the target, we identify the configuration whose bandwidth usage of links in the substrate is as low as possible. With this, VMs are consolidated in the CUs to a larger extent, resulting in significant bandwidth savings.

5) Generation of problem instances: To generate a problem instance, we first select one of the four substrates, *i.e.,* 25, 50, 75, 100 km^2, and one of the three density levels for the network slices, *i.e.,* 33%, 66%, and 100%. For each

of the use cases, we then randomly select the number of VNFs composing the network slice between three (*i.e.,* small network slice) and five (*i.e.,* big network slice). Furthermore, we randomly select the *scaling factor* of each VNF, *i.e.,* the number of VMs with which the VNF is instantiated, between x2, x4, x6, and x8. Note that the first VNF corresponds to a load balancer instantiated with a single VM, as depicted in Figure 4. After the number of VNFs and the scaling factors are decided, the network slices are generated. Finally, we run a simple Integer Linear Programming (ILP) model implemented on a commercial solver[3] to find pairs of source and target configurations. Given the extremely large amount of possible combinations of parameters, we set a time limit of 10 minutes as the solving time for finding each configuration. The main driver for using an ILP to find source and target configurations, instead of a more scalable heuristic, is the simplicity of the formulation and usage of the library.

The ILP is used only to obtain the pairs of source and target configurations, and we do not use it to find the sequence of VM migrations to reconfigure the network slices. Furthermore, getting optimal source and target configurations is not the main goal. Instead, we are interested in finding pairs of feasible configurations that require a non-trivial number of VM migrations to reconfigure the network slices.

E. Characterization and validation

The dataset consists of 241 problem instances. Table IIa illustrates the distribution of the problem instances across all combinations of substrate and use case density. Instances with use case density of 33% are predominant because they comprise a much smaller number of network slices, and thus the search for source and target configurations requires less computing time. Note that we were not able to generate any instance when combining a very large substrate (75-100 km^2) and high use case density (100%).

Table IIb shows the average number of VNFs. Across all combination of parameters, there is a predominance of small network slices, *i.e.,* comprising three VNFs. Instead, Table IIc shows the average number of VMs. Note that the relationship between VNF and VM is *one-to-many* because a VNF can have multiple instances, each one of which corresponds to a different VM. Combinations of parameters that result in simpler problem instances, *e.g.,* substrate of 25 km^2 and use case density of 33%, record a higher average number of VMs. When considering more difficult parameter combinations, we were able to generate source-target configuration pairs only for network slices with a small number of VMs. For example, when considering a substrate of 50 km^2 and a use case density of 100%, *i.e.,* each CU is used simultaneously by all use cases, the average number of VMs per network slice is 9.33. Specifically, the ILP was unable to generate source-target configuration pairs when dealing with higher numbers of VMs.

We define the *lower-bound* of a problem instance as the number of VM migrations that are required to reconfigure network slices from source to target when the substrate is overprovisioned, *i.e.,* hypothetical infinite bandwidth and zero latency on links. The scenario resembles a reconfiguration problem within a data center where bandwidth and latency requirements are less stringent than in 5G. The lower-bound is one indicator of the complexity of a problem instance because finding longer sequences is computationally harder in the general case. Table IId shows the average lower-bound of the problem instances in the dataset. All combinations of parameters record a similar average lower-bound, spacing from 18 to around 28 VM migrations.

As described in §III-D4, each pair of source and target configurations represents a scenario in which resources on the network edge are initially unused, while after the reconfiguration the network is leveraging such edge resources as much as possible. In the following, we assess the degree to which the problem instances represent these scenarios. Table IIIa and Table IIIb compare two metrics computed separately on source and target configurations to illustrate the actual benefits of reconfiguring the network slices. Table IIIa compares the average CPU usage in the CUs. Note that the usage of CUs CPU in target configurations is higher. Observe also that a significant share of the VMs are instantiated in the CUs even in source configurations because of stringent bandwidth and latency requirements, thus justifying the limited difference. On the other hand, Table IIIb compares the average bandwidth usage in the substrate. The reduction in bandwidth usage in the target configurations is outstanding, reducing the bandwidth usage to zero for some combinations of parameters, *i.e.,* all VMs of a network slice are consolidated in the same host.

IV. Migration Sequences with Matryoshka

As shown in Figure 1, state-of-the-art techniques for solving the reconfiguration problem are not suitable for 5G. We therefore design Matryoshka, which addresses the complexity of 5G requirements using different strategies, such as optimizations and speedup improvements through parallelization. Similarly to the work of Dow *et al.* [11], our technique is also based on A* search. In this section, we illustrate the components of Matryoshka, namely (i) the A* search optimizations, (ii) the divide-and-conquer approach, and (iii) the parallelization.

A. A search optimizations*

The optimizations we have designed for A* search are aimed at (i) reducing the search space, and (ii) improving the speed of the steps done by the algorithm. To explain the details of our optimizations, we introduce the concept of a *migration graph*. Given a problem instance, a migration graph consists in a directed graph in which each node represents one of the possible configurations of the network slices and each edge represents a VM migration. For a network with n hosts and a total of k VMs across all network slices, a migration graph has n^k nodes, and each node has $k(n-1)$ incoming edges and $k(n-1)$ outgoing edges. Figure 6 shows an example of a migration graph. *Green nodes* with vertical hatching represent feasible configurations, while *red nodes* with horizontal hatching represent infeasible configurations. Since the source and target configurations correspond to two nodes in the migration graph, solving the problem instance can

[3]IBM CPLEX Optimization Studio 12.7.1

Subst. (km^2.)	Density 33%	66%	100%
25	25.31	2.91	0.83
50	27.39	7.05	0.83
75	17.43	2.07	0.00
100	14.52	1.66	0.00

(a) *% of instances.*

Subst. (km^2.)	Density 33%	66%	100%
25	3.78	3.10	3.00
50	3.70	3.24	3.00
75	3.67	3.27	N/A
100	3.72	3.50	N/A

(b) *Number of VNFs in a network slice (AVG).*

Subst. (km^2.)	Density 33%	66%	100%
25	14.22	9.48	9.67
50	14.28	10.14	9.33
75	13.43	10.07	N/A
100	12.83	10.17	N/A

(c) *Number of VMs in a network slice (AVG)*

Subst. (km^2.)	Density 33%	66%	100%
25	22.18	25.14	21.00
50	25.48	28.12	18.00
75	23.79	26.08	N/A
100	28.29	27.00	N/A

(d) *Reconfiguration problem lower-bound (AVG)*

TABLE II: **Dataset characterization.** *Table IIa shows the composition of the dataset when problem instances are grouped by the coverage area of their substrate and their use case density. For each group, we show the following metrics: the number of VNFs in the network slices (IIb), the number of VMs in the network slices (IIc,) and the problem instance lower-bound, i.e., the minimum number of VM migrations to solve the reconfiguration problem (IId). The values shown correspond to the average values computed over all the problem instances in the group. The darker the cell shade, the closer the value is to its maximum.*

Subst. (km^2.)	Density 33%	66%	100%
25	57.75 → 64.03	80.03 → 83.97	100.0 → 100.0
50	31.66 → 35.31	71.02 → 74.42	96.70 → 98.21
75	31.66 → 35.31	71.02 → 74.42	96.70 → 98.21
100	47.75 → 49.29	77.11 → 78.00	N/A

(a) *AVG edge CPU utilization (%)*

Subst. (km^2.)	Density 33%	66%	100%
25	54.03 → 22.10	79.84 → 29.57	91.94 → 65.32
50	37.12 → 0.00	61.65 → 25.14	64.66 → 22.41
75	33.43 → 0.00	39.66 → 0.00	N/A
100	39.67 → 2.86	60.78 → 29.45	N/A

(b) *AVG bandwidth utilization (%)*

TABLE III: **Comparison between source and target configurations (source → target).** *Reconfiguring from source to target shifts the VMs towards the network edge, thus resulting in higher CPU consumption in the CUs and lower bandwidth utilization within the network. The darker is the cell shade, the bigger is the difference between the values.*

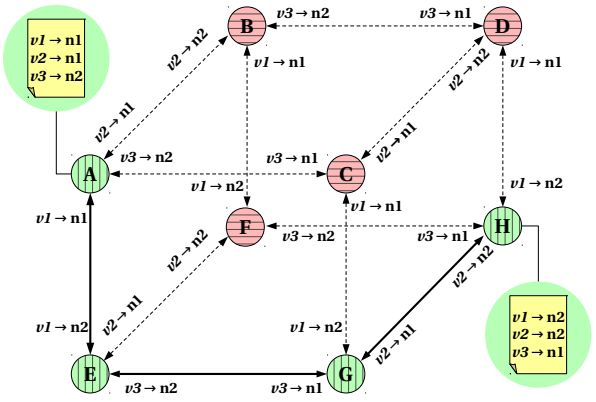


Fig. 6: **Example of a migration graph.** *This migration graph consists of a substrate with two hosts $n1$, $n2$ and a slice with three VMs $v1$, $v2$, $v3$. Each node describes a configuration, and each directed edge describes the migration of a VM to a different host. For example, the edge $v2 \rightarrow n2$ represents the migration of VM $v2$ to host $n2$. Green nodes with vertical hatching represent feasible configurations, while red nodes with horizontal hatching represent infeasible configurations.*

be interpreted as finding a path from source node to target node traversing only green nodes. The sequence of directed edges comprising the path represents a sequence of VM migrations that solves the problem instance.

In the following, we leverage the migration graph to illustrate four optimizations we designed to speed up A* search.

1) Reducing search space: When discovering new nodes, A* search assigns a low priority to red nodes because we are interested in finding paths comprising green nodes. Nevertheless, storing a red node and its priority in the openset is (i) not scalable because there is an exponential number of red nodes in a migration graph, and (ii) useless because we do not accept a path comprising a red node as a solution for the problem instance. We tuned A* search to discard red nodes as soon as they are discovered, thus gaining benefits in memory occupied and speed of reads/writes in openset. Avoiding red nodes ensures that neither SLOs are violated nor is the network overloaded during reconfiguration. The red nodes represent infeasible configurations, therefore in this context discarding red nodes is not just a simple optimization. Instead, it hides a portion of the search space that would eventually be discarded.

2) Early pruning: Given a problem instance, there can be pairs of VMs for which the SLOs enforce them to be always instantiated in the same host. For example, the SLOs may require that the latency between the two VMs must be less than 1 ms, and the substrate may not have any links with a latency lower than 1 ms. Before starting A* search, we pre-process the network slices to identify pairs of VMs falling in this category. For all such pairs, we tune A* search to discard the edges representing the possible migrations of any of the two VMs. This significantly reduces the out-degree of the nodes in the migration graph, thus speeding up the exploration.

3) Reducing scope of feasibility checks: Assigning a color to a node requires checking all the SLOs: if the configuration of the node incurs one or more violations, then the node is red, otherwise it is green. When exploring a neighbor of a green node instead, it is sufficient to check the SLOs related with the VM and host of the migration leading to the neighbor. For example, suppose we are examining a neighbor reached by migrating VM $v1$ to host $n1$. We can limit the check to the SLOs dealing with (i) CPU and memory of host $n1$, and (ii) bandwidth and latency of the virtual links of $v1$. If the SLOs are not violated, then the node is green because the

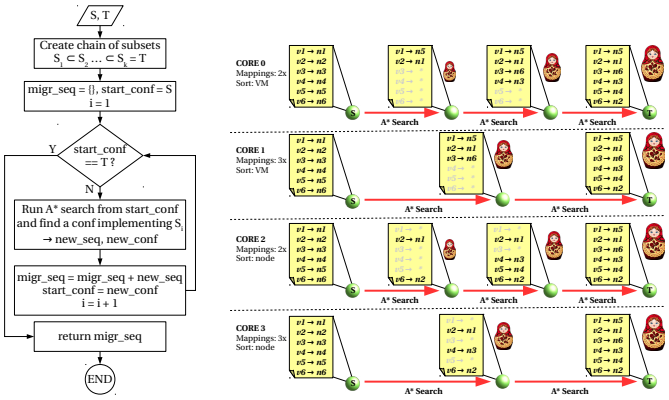


Fig. 7: **Divide-and-conquer to reconfigure from source S to target T.** *Each $v \rightarrow n$ represents a mapping of VM v to host n. We use multiple strategies to create different chains of subsets of the mappings of the target configuration. For each chain, we consider each subset starting from the smallest one, and we iteratively use A* search to find a sequence of VM migrations leading to a feasible configuration that implements the mappings of the subset. The different chains are processed by different cores simultaneously.*

node from which we arrived is also green and therefore all the other SLOs are met. Reducing the scope of the feasibility check saves time during the graph exploration.

4) Red nodes cache: The high in-degree and out-degree of nodes in a migration graph makes A* search more likely to visit a single node multiple times during the exploration. This can result in a waste of time because the configuration of a node is checked every time the node is visited. To address this issue, we introduced a cache for red nodes. Every time a red node is detected, a signature of its configuration, *e.g.,* the value returned by a hash function, is stored in the cache. Subsequently, on discovering a new node, we check its signature in the cache. If we get a cache hit, then its configuration is infeasible and the node can be discarded. If we get a cache miss instead, then we proceed with checking the feasibility of the configuration.

B. Divide-and-conquer approach

We define a *mapping* as an indication of which host is instantiating a certain VM. For example, a mapping $v1 \rightarrow n1$ indicates that VM $v1$ is instantiated at host $n1$. A configuration thus consists of a collection of mappings, one for each of the VMs of the network slices. On each VM migration, the configuration changes the mapping of the migrated VM to the new host.

The original version of A* search, which we call *plain* A* search, stops only when it finds the node whose configuration corresponds to the target configuration, *i.e.,* the mappings of the configuration are all equal to the ones of the target configuration. Instead, our divide-and-conquer approach proceeds as follows. From the mappings of the target configuration, we create a *chain of subsets* $S_1, S_2, \ldots S_k$ such that $\forall i < j, S_i \subset S_j$, *i.e.,* the mappings in S_i are in S_j. We then consider the subsets one after the other, starting from the smallest one: we run A* search to find a feasible configuration which implements the mappings contained into the subset. Once we find one such configuration, we record the sequence of VM migrations leading to the configuration, we select the next subset of mappings in the chain, and we run A* search again starting from the latest configuration found. The process is repeated till the end of the chain of subsets. The last run of A* search will reach the target configuration because the last subset corresponds to the entire set of mappings. Finally, we join the sequences of VM migrations obtained by the several runs of A* search, thus obtaining a sequence of VM migrations reconfiguring from source to target.

We present an example of our divide-and-conquer approach in Figure 7. The chain of subsets in this example is S_1, S_2, and S_3, where S_3 contains all the mappings of the target configuration, $S_2 = \{v1 \rightarrow n5, v2 \rightarrow n1, v3 \rightarrow n6, v4 \rightarrow n3\}$, and $S_1 = \{v1 \rightarrow n5, v2 \rightarrow n1\}$. We consider S_1 first, and we run A* search from the source configuration to find a feasible configuration that implements the mappings in S_1. This takes advantage of the small number of mappings, and the existence of many configurations meeting these requirements, allowing A* search to stop at the first configuration found in the exploration. When one such configuration is found, the sequence of VM migrations leading to the configuration is saved.

The subsequent iteration considers the mappings in S_2 instead, and it runs A* search starting from the configuration found in the previous iteration. The same applies in the last iteration, where we consider the mappings in S_3. Also the problems in these iterations are easier than the problem in plain A* search because we run A* search starting from partial solutions, *i.e.,* configurations that already implement a subset of the mappings in the target configuration. At the end of the last iteration, we join together the sequences of VM migrations obtained in the different iterations. In this way, we obtain a sequence of VM migrations reconfiguring from source to target.

C. Parallelization

There are many ways of creating chains of subsets when adopting the divide-and-conquer approach. We define a *strategy* as a set of criteria used to indicate *how* and *how many* mappings to select for creating a chain of subsets. For example, in Figure 7 we can see four different strategies being used for creating chains of subsets, by varying how mappings are selected for each subset (sorting by VM or by destination node in the substrate) and the difference in the number of mappings between consecutive subsets (2 or 3).

Unfortunately there is no clear hint on why one strategy should perform better over the others. Therefore, we leverage the multicore capabilities of modern computers to process the different chains of subsets in parallel. For example, in Figure 7 the four chains of subsets are processed on different cores. Matryoshka organizes the parallel processing of different chains of subsets in two different modes: *independent mode* and *sharing mode*. In both modes, each core handles a chain of subsets created by a different strategy.

1) Independent mode: In this mode, each core processes its own chain of subsets independently. Matryoshka simply returns the sequence of VM migrations returned by the fastest core to terminate the processing of the chain of subsets.

2) Sharing mode: In this mode, Matryoshka stops the processing in all the cores when one of them provides the sequence of VM migrations for the *first subset* in the chain of subsets. At this point, Matryoshka re-creates the chains of subsets using the same strategies but using the configuration found as the new source configuration. Then, it launches the processing of the new chains on the cores, repeating the loop until it finds the sequence from the source to the target.

V. Evaluation

We implemented state-of-the-art A* search and Matryoshka using Python 3.5. In this section, we use the dataset described in §III to evaluate Matryoshka and compare it with state-of-the-art A* search. Our dataset consists of 241 problem instances having different values of coverage area of the substrate and density of use cases, the details of which are in Table IIa. Henceforth, we use the expression *dataset instances* to refer to the set of 241 problem instances of the dataset. In the following, we define the research questions of our evaluation, we provide a general description of the tests conducted, and finally we show the results.

A. Research questions

The evaluation answers the following three questions:

Question a) *Is Matryoshka able to solve more problem instances than state-of-the-art A* search in a limited time window?* Matryoshka was designed because state-of-the-art techniques are unlikely to find a VM migration sequence for reconfiguring 5G network slices. One aim of the evaluation is to test whether Matryoshka meets its design goals.

Question b) *What is the quality of the solutions provided by Matryoshka?* Given the subdivision into sub-problems, Matryoshka is not guaranteed to find the shortest sequence of VM migrations. The aim is to assess the degradation in quality of the solutions, in terms of additional VM migrations required compared to the state of the art and also the theoretical lower-bound. Note that this lower-bound might represent migration sequences that include infeasible configurations.

Question c) *Which instances are more difficult to solve for Matryoshka?* The problem instances having a large substrate area and a high use case density are expected to be hard to solve. We corroborate this by examining the time required by Matryoshka to solve the problem instances in our dataset.

B. Methodology

To answer the research questions, we run the state-of-the-art A* search and Matryoshka on the problem instances of the dataset. In particular, we run both Matryoshka in independent mode and Matryoshka in sharing mode. We allow up to 60 minutes of solving time for each problem instance. Although this time limit is high for realistic scenarios, we use it to highlight the strengths and weaknesses of the approaches along with the hardness of the problem instances.

We generate the strategies used by Matryoshka varying two parameters: (i) the number of mappings considered simultaneously, and (ii) the criterion for the selection of the mappings at each iteration. For the number of mappings, we consider strategies varying from a single mapping to five mappings at a time. For selection we use the following two criteria. Using the first criterion, we consider one network slice at a time: we first select the migrations involving VMs of a certain network slice, and we select other migrations after all the migrations of that slice are completed. Using the second criterion, we consider a destination host at a time. In this case, we first select all the migrations whose VMs need to be reconfigured to a given destination host before selecting other migrations. The advantage of adopting these two criteria is that they allow obtaining measurable achievements within the reconfiguration process, *e.g.,* two network slices reconfigured over four. Both criteria sort the mappings in ascending order using an internal identifier, for example using the first criterion the network slices are sorted using a numeric id. To mitigate any bias coming from this feature, we consider also a variant of the two criteria in which we pick the mappings in descending order, resulting in a total of four criteria. Finally, we generate the strategies by performing all the possible combinations of the values of the two parameters, for a total of 20 strategies obtained. Note that we tune the machine on which we run the evaluation to allocate 20 cores to Matryoshka to avoid slowdowns due to core contention between different strategies.

To answer our research questions, we proceed as follows:

Question a) We measure the time required to solve each problem instance. We then compute the Cumulative Distribution Function (CDF) of the number of problem instances that are solved over time.

Question b) During our evaluation, we observed that Matryoshka was able to solve more problem instances than the state-of-the-art A* search. We therefore consider two subsets of problem instances, S1 and S2, where S1 contains the problem instances solved by all techniques while S2 contains problem instances solved only by Matryoshka in independent mode and sharing mode. For each problem instance in S1 and S2, we use the lower-bound on the number of VM migrations required as the baseline. We then compare the length of the sequence of VM migrations found by the techniques evaluated against this baseline. Specifically, we look at the absolute (ABS) and relative (REL) difference in the length of the sequences.

Question c) We consider the problem instances in subset S2. For both Matryoshka modes, we group the problem instances by substrate area and use case density, and we compute the average of the recorded solving times.

C. Results

In the following, we present the results and answer each of our research questions. For brevity, we use SOTA to refer to the state-of-the-art A* search described in the work of Dow *et al.* [11]. Similarly, we use MAT-I and MAT-S to refer to Matryoshka in independent and sharing modes respectively.

Question a) In Figure 8, we present the distribution of the completion time for each of the techniques, and we

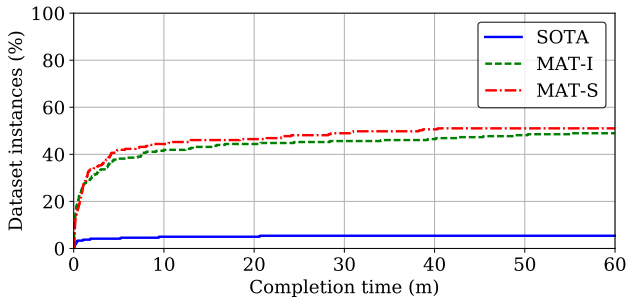
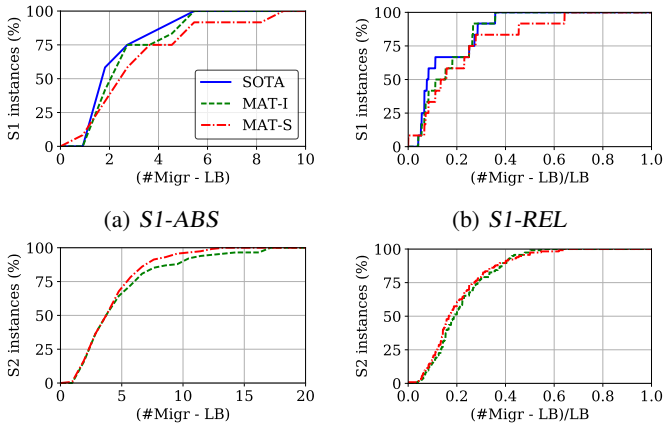


Fig. 8: **CDF of the number of problem instances solved.** *MAT-I and MAT-S are able to solve 8x more problem instances than SOTA within the first 10 minutes of solving time.*



(a) *S1-ABS* (b) *S1-REL*

(c) *S2-ABS* (d) *S2-REL*

Fig. 9: **Differences between number of VM migrations given by the reconfiguration technique (#Migr) and lower-bound (LB).** *The first column (Fig. 9a and Fig. 9c) shows the CDF of absolute differences (ABS). The second column (Fig. 9b and Fig. 9d) shows the CDF of relative differences (REL), i.e., absolute difference divided by LB. In the first row (Fig. 9a and Fig. 9b), the CDFs are computed over the problem instances solved by all three techniques (S1). In the second row (Fig. 9c and Fig. 9d), the CDFs are computed over all problem instances solved by both MAT-I and MAT-S (S2). MAT-I and MAT-S add a maximum of five VM migrations in the majority of the cases, which is in line with the performance of SOTA.*

observe that Matryoshka is more likely to find a VM migration sequence compared to SOTA. Specifically, SOTA requires more than 60 minutes for approximately 95% of the problem instances. As described in Section I, we can conclude that SOTA is not a suitable technique for reconfiguring network slices. Instead, both MAT-I and MAT-S are able to solve more than 40% of the problem instances already after 10 minutes of execution. After the whole 60-minute time window, MAT-I and MAT-S are able to solve around half of the problem instances of the dataset. Please note that MAT-S performs slightly better than MAT-I, which confirms the theoretical advantages of the sharing mode over the independent mode.

Question b) Figure 9 illustrates the CDFs computed over ABS and REL. For the subset of problem instances in S1, Figure 9a shows that MAT-I incurs a maximum of five additional VM migrations, which is in line with SOTA. We also observe that the behavior of MAT-S is similar to that of SOTA for approximately 90% of the problem instances. Figure 9b shows that both MAT-I and SOTA incur up to 30% additional VM migrations. Instead, MAT-S incurs an overhead of up to 60-70% over the lower-bound. This hints that the benefits brought by MAT-S in terms of number of problem instances solved are counter-balanced by a general lower quality of the solutions.

Focusing on the results of subset S2, we can see that MAT-I and MAT-S record very similar performance. Figure 9c shows that both techniques incur an overhead of less than ten VM migrations in the majority of the cases. Figure 9d shows that the overhead corresponds to less than 50% of the lower-bound for the greatest part of the problem instances solved.

Question c) Table IVa shows the average solving time of the problem instances solved by MAT-I grouped by substrate area and use case density. As expected, the bigger substrates have higher running time. Surprisingly instead, the increase in the use case density brings a reduction of the average solving time. We can justify this unexpected outcome by considering that (i) the number of problem instances generated with higher use case density is smaller and (ii) the few such problem instances that have been solved record a short solving time. This suggests that the higher the use case density, the easier it is to identify two extreme groups of problem instances. Indeed, each problem instance is either computationally easy to solve, or it requires more than 60 minutes of execution.

Table IVb shows the same statistics for MAT-S. In the case of MAT-S, the collected data is fully aligned with common sense, as bigger substrates and higher use case density result in higher average solving time. Please note that in both Table IVa and Table IVb the number of samples over which each average is computed varies significantly.

VI. Discussion

a) Optimality criterion: When running A* search, Matryoshka assigns a unitary cost to each VM migration incurred, thus finding sequences comprising the fewest VM migrations. This choice reflects the aim of reconfiguring the network slices as quickly as possible. We argue that an operator could be interested in other aims as well, such as finding a solution whose VM migrations do not occur on leased lines. In this case, the cost of each VM migration should reflect how much the migration is aligned with the objective, *e.g.*, how much the migration uses leased lines. We plan to extend Matryoshka to support customized objectives by developing an algorithm to produce and assign costs according to the objectives in input.

b) Other domains: State-of-the-art reconfiguration techniques fail in 5G contexts also because they are designed for data center networks, in which latency requirements of applications are typically not critical. The advent of AR devices such as Google Glass and Microsoft HoloLens brings a new set of latency-sensitive applications to data center networks [35], thus making past techniques likely to fail in this domain too. Despite being designed for 5G networks, Matryoshka can be easily extended to also reconfigure generic virtual networks for latency-sensitive applications deployed in data centers.

Substrate (km^2)	Density 33%	Density 66%
25	113.35 (34)	9.11 (2)
50	287.42 (38)	237.90 (7)
75	571.54 (28)	416.70 (2)
100	1168.53 (6)	598.68 (1)

(a) *MAT-I*

Substrate (km^2)	Density 33%	Density 66%
25	176.20 (36)	10.90 (2)
50	191.94 (37)	262.60 (7)
75	309.50 (29)	339.51 (2)
100	803.25 (8)	1082.69 (2)

(b) *MAT-S*

TABLE IV: **Characterization of problem instances solved by Matryoshka.** *Matryoshka is most effective in solving problem instances with low use case density (33%). The tables also show the AVG completion time(s) of MAT-I and MAT-S. The number of samples over which each AVG value has been computed is indicated in parentheses. The columns related to 100% density are missing because none of the corresponding problem instances has been solved.*

c) Limitations: At the current stage, Matryoshka does not have bounds on the time required to solve a problem instance. In addition, Matryoshka does not have bounds on the distance from optimality of the solutions provided. Despite the encouraging results on both solving time and quality of solutions, operators need such bounds to better define the contexts in which Matryoshka can or cannot be used. Finding such bounds is a top priority in our research agenda.

Lastly, Figure 8 shows that around 50% of problem instances are unsolved even considering the whole 60-minutes time window. Matryoshka thus represents just the first step in the direction of finding a complete solution to the reconfiguration problem. We hope the dataset will stimulate further research on the problem, possibly leading to the development of even faster techniques.

d) Management of State and Violation of SLOs: Virtualized Infrastructure Managers (VIMs) allow network operators to launch, migrate, and tear down VMs. By monitoring and managing the life cycle of the virtual resources, these platforms assist in managing the *state* of the network resources. They also perform basic actions in response to pre-defined triggers, *e.g.*, scale out when VM CPU ready exceeds a 5% threshold [33]. Knowing how to reconfigure the entire network without violating SLOs can complement their capabilities. Matryoshka addresses this by providing a sequence of such actions for reconfiguring network slices.

SLOs are known to be stringent [4], and violating them can result in monetary losses for network operators. Nevertheless, some researchers have proposed ways of managing occasional SLO violations in network slicing that might come from overbooking the substrate [36]. In contrast, Matryoshka looks for a solution that does not violate any SLOs.

VII. Related Work

a) Modeling Datasets: When data from real deployments is scarce, researchers usually leverage mathematical modeling [32], simulation techniques or both [37]; or simply obfuscate the data in cases where confidentiality agreements apply [36]. The first approaches can lead to inaccurate results given the complexity added by the multiple layers of network virtualization, whereas the second approach is typically not reproducible. To overcome these issues, researchers have created shared dataset collections. For example, SNDlib [38] and the Internet Topology Zoo [39] collect topologies of real networks, while Crawdad [40] collects traces from users of various wireless technologies. Unfortunately, we have not found available public datasets about 5G networks at scale, which justify the modeling dataset contributed in Section III.

b) Reconfiguration: In the past years, several papers have investigated how to reconfigure virtual networks by migrating VMs. Most of these propose a heuristic on how to order the VM migrations. The work of Li *et al.* [41] prioritizes the migration of the VMs whose page dirtying rate is higher. Sarker *et al.* [8] propose to migrate the "less chatty" VMs first. The work of Bari *et al.* [9] prioritizes the migrations that result in more available bandwidth. A similar idea is developed also in the work of Lu *et al.* [42]. More aligned with Matryoshka, other works focus on avoiding, or minimizing, the SLO violations, such as the work of Ghorbani *et al.* [43], the work of Al-Haj *et al.* [44], and the already-mentioned work of Dow *et al.* [11]. Nevertheless, these works focus on VM requirements such as CPU, memory, and bandwidth, thus neglecting latency requirements spanning several VMs simultaneously. Thus, they are inadequate for reconfiguring network slices in 5G networks.

c) Network slicing: Despite not being commercialized yet, network slicing has already received significant attention from the research community [45]. More precisely, researchers have focused on how to *configure* network slices so far. Indeed, the majority of the work focuses on both theoretic and technical aspects of resource management, *i.e.*, allocating the virtual resources of the network slices on the shared substrate. The work of Samdanis *et al.* [46] proposes a 5G network slices broker, which takes care of configuring the network slices on the physical substrate. Network slice requirements also include a time duration, after which the corresponding virtual resources are deallocated from the network. The works of Zhang *et al.* [47] and Ksentini *et al.* [48] focus on resource reservation in the Radio Access Network. The first work proposes a model to allocate radio resources maximizing link capacity, while the second work focuses on the mechanisms to enforce resource allocation decisions. Finally, the work of Salvat *et al.* [36] proposes the principles of overbooking in network slices resource allocation to maximize the revenue of the operator. While these works focus on how to *configure* network slices, *i.e.*, finding an optimal placement of virtual resources in the network, to the best of our knowledge our work represents the first effort on studying how to *reconfigure* network slices, finding a sequence of migrations that changes the current placement of virtual resources to a new one.

VIII. Conclusions

We proposed Matryoshka, a divide-and-conquer technique to find a sequence of VM migrations to reconfigure 5G network slices. Given that 5G networks are in their early stages of deployment, we collected technical details from a variety of sources to create a dataset of 5G scenarios. We trust that our modeling dataset will spark research activities towards solving more complex 5G scenarios. We then used the dataset to compare Matryoshka with state-of-the-art techniques. Our results show that Matryoshka clearly outperforms state-of-the-art techniques by being 10 times more effective in finding good quality migration sequences. We believe that our divide-and-conquer approach can be beneficial also in other contexts in which A* search still represents the state of the art.

In the future we plan to develop mathematical bounds for the completion time of Matryoshka and also for the quality of its output to assess its effectiveness theoretically. Moreover, we plan to integrate our technique into network slices orchestration platforms to realize a unique system offering both configuration and reconfiguration of network slices.

IX. Acknowledgement

This work has been supported by Nokia Bell Labs. The work is in part supported by the Nokia Center for Advanced Research (NCAR) and the 5G FORCE research project.

References

[1] 3rd Generation Partnership Project (3GPP), "Service requirements for next generation new services and markets," TS 22.261, Sep. 2018.

[2] S. Elayoubi, M. Fallgren, P. Spapis, G. Zimmermann, D. Martín-Sacristán, C. Yang, S. Jeux, P. Agyapong, L. M. Campoy, Y. Qi, and S. Singh, "5g service requirements and operational use cases: Analysis and METIS II vision," in *European Conference on Networks and Communications, EuCNC 2016, Athens, Greece, June 27-30, 2016*, 2016, pp. 158–162.

[3] Next Generation Mobile Networks (NGMN), "Description of Network Slicing Concept," https://www.ngmn.org/wp-content/uploads/160113_NGMN_Network_Slicing_v1_0.pdf, Jan 2016.

[4] D. Lugones, J. A. Aroca, Y. Jin, A. Sala, and V. Hilt, "Aidops: a data-driven provisioning of high-availability services in cloud," in *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*, 2017, pp. 466–478.

[5] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 275–287.

[6] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the Virtual Library Operating System," *Queue*, vol. 11, no. 11, pp. 30:30–30:44, Dec. 2013.

[7] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, 2017.

[8] T. K. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *2013 IEEE International Conference on Granular Computing, GrC 2013, Beijing, China, December 13-15, 2013*, 2013, pp. 253–258.

[9] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "CQNCR: optimal VM migration planning in cloud data centers," in *2014 IFIP Networking Conference, Trondheim, Norway, June 2-4, 2014*, 2014, pp. 1–9.

[10] J. Liu, L. Su, Y. Jin, Y. Li, D. Jin, and L. Zeng, "Optimal VM migration planning for data centers," in *IEEE Global Communications Conference, GLOBECOM 2014, Austin, TX, USA, December 8-12, 2014*, 2014, pp. 2332–2337.

[11] E. M. Dow and J. N. Matthews, "WAYFINDER: parallel virtual machine reallocation through a* search," *Memetic Computing*, vol. 8, no. 4, pp. 255–267, 2016.

[12] 3rd Generation Partnership Project (3GPP), "System architecture for the 5G System (5GS)," TS 23.501, Dec. 2018.

[13] Internet Engineering Task Force (IETF), "Service Function Chaining Use Cases in Mobile Networks," Internet Draft, May 2018.

[14] F. Hao, M. S. Kodialam, T. V. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 238–249, 2017.

[15] S. Asif, *5G Mobile Communications: Concepts and Technologies*. CRC Press, 2018.

[16] Zakaria Tayq, "Fronthaul integration and monitoring in 5G networks," Doctoral Dissertation, University of Limoges, https://tel.archives-ouvertes.fr/tel-01708493/document, Dec 2017.

[17] Cisco, "Cisco data center infrastructure 2.5 design guide," https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.html, 2011, last access: Feb 2020.

[18] 5G-XHaul, "Network Topology Definition," Deliverable 2.3, Jan. 2017.

[19] 3rd Generation Partnership Project (3GPP), "NG-RAN; Architecture description," TS 38.401, Sep. 2018.

[20] International Telecommunication Union (ITU), "Transport network support of IMT-2020/5G," TR, Feb. 2018.

[21] European Telecommunications Standards Institute (ETSI), "MEC in 5G networks," WP 28, Jun. 2018.

[22] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[23] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for Mobile Networks - A Technology Overview," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 405–426, 2015.

[24] Nokia, "Nokia AirFrame open edge server," https://onestore.nokia.com/asset/205107, last access: Feb 2020.

[25] Cisco, "A Network Infrastructure for the Future with 5G - Introducing the NCS 500 Series of Routers," https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/0306-msn-ckn.pdf, last access: Oct 2018.

[26] Nokia, "Microwave Network Evolution - Software Defined Networking and a Layer 3 VPN Vision," https://pages.nokia.com/T00275.5G.MW.white.paper.html, last access: Feb 2020.

[27] Netmanias, "Survey on 5G RAN Practical Deployment Scenario: From 2-tier (4G) to 3-tier (5G)," https://www.netmanias.com/en/post/reports/13103/5g-c-ran-fronthaul/brief-survey-on-5g-ran-practical-deployment-scenario-from-2-tier-4g-to-3-tier-5g, Jan. 2018, last access: Feb 2020.

[28] M. Jaber, M. A. Imran, R. Tafazolli, and A. Tukmanov, "5g backhaul challenges and emerging research directions: A survey," *IEEE Access*, vol. 4, pp. 1743–1766, 2016.

[29] D. Sattar and A. Matrawy, "An empirical model of packet processing delay of the open vswitch," *CoRR*, vol. abs/1706.06631, 2017.

[30] VMWare, "Network I/O Latency on VMware vSphere®5," https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/network-io-latency-perf-vsphere5-white-paper.pdf, last access: Feb 2020.

[31] Next Generation Mobile Networks (NGMN), "5G White Paper," https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, Feb 2015.

[32] C. Fuerst, S. Schmid, P. L. Suresh, and P. Costa, "Kraken: Online and elastic resource reservations for cloud datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 422–435, 2018.

[33] VMware, "Performance Troubleshooting - CPU Ready Time," https://learnvmware.online/2018/03/08/performance-troubleshooting-cpu-ready-time/, Mar. 2018, last access: Feb 2020.

[34] International Telecommunication Union (ITU), "Framework and overall objectives of the future development of IMT for 2020 and beyond," Tech. Rep. M.2083-0, Sep. 2015.

[35] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. L. Klatzky, D. P. Siewiorek, and M. Satyanarayanan, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose / Silicon Valley, SEC 2017, CA, USA, October 12-14, 2017*, 2017, pp. 14:1–14:14.

[36] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 353–365.
[37] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.*, ser. Wiley professional computing. Wiley, 1991.
[38] "SNDlib - library of test instances for Survivable fixed telecommunication Network Design," http://sndlib.zib.de/home.action, 2006.
[39] "The Internet Topology Zoo," http://www.topology-zoo.org/.
[40] "CRAWDAD - Community Resource for Archiving Wireless Data At Dartmouth," https://crawdad.org/.
[41] X. Li, Q. He, J. Chen, K. Ye, and T. Yin, "Informed live migration strategies of virtual machines for cluster load balancing," in *Network and Parallel Computing - 8th IFIP International Conference, NPC 2011, Changsha, China, October 21-23, 2011. Proceedings*, 2011, pp. 111–122.
[42] T. Lu, M. Stuart, K. Tang, and X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live migration," in *9th IEEE International Conference on Networking, Architecture, and Storage, NAS 2014, Tianjin, China, August 6-8, 2014*, 2014, pp. 216–225.
[43] S. Ghorbani and M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees," in *Proceedings of the first workshop on Hot topics in software defined networks, HotSDN@SIGCOMM 2012, Helsinki, Finland, August 13, 2012*, 2012, pp. 67–72.
[44] S. Al-Haj and E. Al-Shaer, "A formal approach for virtual machine migration planning," in *Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, Zurich, Switzerland, October 14-18, 2013*, 2013, pp. 51–58.
[45] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.
[46] K. Samdanis, X. Costa-Pérez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5g network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.
[47] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung, "Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, 2017.
[48] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on RAN: flexibility and resources abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

APPENDIX

Computing Coverage Area of CU: We sum download and upload bandwidth requirements of each macro use case. eMBB: 750 + 125 = 875 Gbps/km^2; URLLC: 10 + 10 = 20 Gbps/km^2; mMTC: 100 + 100 = 200 Gbps/km^2. Then, we obtain the number of VNF instances required for signal processing dividing the bandwidth values by the capacity of a VNF instance for signal processing, *i.e.,* 160 Gbps. eMBB: $\lceil 875/160 \rceil$ = 6 VNF instances/km^2; URLLC: $\lceil 20/160 \rceil$ = 1 VNF instance/km^2; mMTC: $\lceil 200/160 \rceil$ = 2 VNF instances/km^2. Every CU is equipped with a single Nokia AirFrame [24] for traffic processing, which is equipped with 28 cores. Since each VNF instance requires 4 CPU cores, each CU can host 7 VNF instances. Finally, we obtain the coverage area of a single CU for each macro use case. eMBB: 7 / 6 = 1.17 km^2; URLLC: 7 / 1 = 7 km^2; mMTC: 7 / 2 = 3.5 km^2. We now take the average of the three coverage areas to get a single coverage value for a CU. (1.17 + 7 + 3.5) / 3 = 3.89 $\approx$ 4 km^2.