Joint Planning of Network Slicing and Mobile Edge Computing: Models and Algorithms

Bin Xiang, Jocelyne Elias, Fabio Martignon, and Elisabetta Di Nitto

Abstract—Multi-access Edge Computing (MEC) facilitates the deployment of critical applications with stringent QoS requirements, latency in particular. This paper considers the problem of jointly planning the availability of computational resources at the edge, the slicing of mobile network and edge computation resources, and the routing of heterogeneous traffic types to the various slices. These aspects are intertwined and must be addressed together to provide the desired QoS to all mobile users and traffic types still keeping costs under control. We formulate our problem as a mixed-integer nonlinear program (MINLP) and we define a heuristic, named Neighbor Exploration and Sequential Fixing (NESF), to facilitate the solution of the problem. The approach allows network operators to fine tune the network operation cost and the total latency experienced by users. We evaluate the performance of the proposed model and heuristic against two natural greedy approaches. We show the impact of the variation of all the considered parameters (viz., different types of traffic, tolerable latency, network topology and bandwidth, computation and link capacity) on the defined model. Numerical results demonstrate that NESF is very effective, achieving near-optimal planning and resource allocation solutions in a very short computing time even for large-scale network scenarios.

Index Terms—Edge computing, network planning, node placement, network slicing, joint allocation.

1 Introduction

Next generation mobile networks aim to meet different users' Quality of Service (QoS) requirements in several demanding application scenarios and use cases. Among the others, controlling *latency* is certainly one of the key QoS requirements that mobile operators have to deal with. In fact, the classification devised by the International Telecommunications Union-Radio communication Sector (ITU-R), shows that mission-critical services depend on strong latency constraints. For example, in some use cases (e.g., autonomous driving), the tolerable latency is expected to reach less than 1 ms [1].

To address such constraints various ingredients are emerging. First of all, through *Network Slicing*, the physical network infrastructure can be split into several isolated logical networks, each dedicated to applications with specific latency requirements, thus enabling an efficient and dynamic use of network resources [2].

Second, *Multi-access Edge Computing (MEC)* provides an IT service environment and cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network and in close proximity to mobile subscribers [3]. Through this approach, the latency experienced by mobile users can be consistently reduced. However, the computation power that can be offered by an edge cloud is quite limited in comparison with a remote cloud. Fortunately, this problem can be addressed by enabling cooperation among multiple edge clouds, scenario that can be realized in next-generation mobile networks (5G and beyond) as they will be likely built in an ultra-dense manner, where the edge clouds attached to base stations will also be massively deployed and connected to each other in a specific topology.

In this line, we study the case of a complex network organized in multiple *edge clouds*, each of which may be connected to the Radio Access Network of a certain location. All such edge clouds are connected through an arbitrary topology. This way, each edge cloud can serve end user traffic by relying not only on its own resources, but also offloading some traffic to its neighbors when needed. We specifically consider multiple classes of traffic and corresponding requirements, including voice, video, web, among others. For every class of traffic incoming from the corresponding Radio Access Network, the edge cloud decides whether to serve it or offload it to some other edge cloud. This decision depends on the QoS requirements associated to the specific class of traffic and on the current status of the edge cloud.

Our main objective is to ensure that the infrastructure is able to serve all possible types of traffic within the boundaries of their QoS requirements and of the available resources.

In this work, therefore, we propose a complete approach, named *Joint Planning and Slicing of mobile Network and edge Computation resources* (JPSNC), which solves the problem of *operating* cost-efficient edge networks. The approach jointly takes into account the overall budget that the operator uses in order to allocate and operate computing capabilities in its edge network, and *allocates resources*, aiming at minimizing the network operation cost and the total traffic latency of transmitting, outsourcing and processing user traffic, under the constraint of user tolerable latency for each class of

- *B. Xiang and E. Di Nitto are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy, 20133. E-mail: {bin.xiang, elisabetta.dinitto}@polimi.it.*
- *J. Elias is with the Department of Computer Science and Engineering (DISI), University of Bologna, Bologna, Italy, 40126. E-mail: jocelyne.elias@unibo.it.*
- *F. Martignon is with the Department of Management, Information and Production Engineering, University of Bergamo, Bergamo, Italy, 24044. E-mail: fabio.martignon@unibg.it.*

Manuscript received XXX XX, 20XX; revised XXX XX, 20XX.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3107022, IEEE Transactions on Cloud Computing

traffic.

This turns out to be a mixed-integer nonlinear programming (MINLP) optimization problem, which is an $\mathcal{NP}$-hard problem [4]. To tackle this challenge, we transform it into an equivalent mixed-integer quadratically constrained programming (MIQCP) problem, which can be solved more efficiently through the Branch and Bound method. Based on this reformulation, we further propose an effective heuristic, named *Neighbor Exploration and Sequential Fixing (NESF)*, that permits to obtain near-optimal solutions in a very short computing time, even for the large-scale scenarios we considered in our numerical analysis. Furthermore, we propose two simple heuristics, based on a greedy approach. They provide benchmarks for our algorithms, obtain (slightly) sub-optimal solutions with respect to NESF, and are still very fast. Finally, we systematically analyze and discuss with a thorough numerical evaluation the impact of all considered parameters (viz. the overall planning budget of the operator, different types of traffic, tolerable latency, network topology and bandwidth, computation and link capacity) on the optimal and approximate solutions obtained from our proposed model and heuristics. Numerical results demonstrate that our proposed model and heuristics can provide very efficient resource allocation and network planning solution for multiple edge networks.

This work takes the root from a previous paper [5] where we focused exclusively on minimizing the latency of traffic in a hierarchical network, keeping the network and computation capacity fixed. In this paper, we have completely revised our optimization model to cope with a joint network planning, slicing and edge computing problem, aimed at minimizing both the total latency and operation cost for arbitrary network topologies.

The remainder of this paper is organized as follows. Section 2 introduces the network system architecture we consider. Section 3 provides an intuitive overview of the proposed approach by using a simple example. Section 4 illustrates the proposed mathematical model and Section 5 the heuristics. Section 6 discusses numerical results in a set of typical network topologies and scenarios. Section 7 discusses related work. Finally, Section 8 concludes the paper.

2 System Architecture

Figure 1 illustrates our reference network architecture. We consider an edge network composed of *Edge Nodes*. Each of such nodes can be equipped with any of the following three capabilities:

- the ability of acquiring traffic from mobile devices through the Remote Radio Head (RRH), such nodes are those we call *Ingress Nodes*;
- the ability of executing network or application level services requiring computational power, this is done thanks to the availability of an *Edge Cloud* on the node;
- the ability to route traffic to other nodes.

Not all nodes must have all the three capabilities, so, in this respect, the edge network can be constituted of heterogeneous nodes.

Each link (i, j) between any two edge nodes, i and j, has a fixed bandwidth, denoted by B_{ij}. Each *Ingress Node* k has a specific ingress network capacity C_k, which is a measure of its ability to accept traffic incoming from mobile devices. Nodes able to perform some computation have a computation capacity S_i. One of the objectives of the planning model presented in this paper is to determine the optimal value of the computation capacity that must be made available at each node.

We assume that users' incoming data in each Ingress Node is aggregated according to the corresponding *traffic type* $n \in \mathcal{N}$. Examples of traffic types can be video, game, data from sensors, and the like. They group demands or services having the same requirements. We assume that the network has a set of slices of different types and that each slice aggregates traffic of the same type. Therefore, all demands or services in the same slice could be treated in the same manner and could share network resources in a soft way like the concept of *soft slicing* introduced in [6]. Our slicing model is also similar in part to the one introduced in [7], where the authors assume that mobile subscribers consume a variety of heterogeneous services and the operator owning the infrastructure implements a set of slices where *each slice* is dedicated to a *different subset of services*.

In Figure 1 traffic of different types is shown as arrows of different colors. From each Ingress Node, traffic can be split and processed on all edge clouds in the network; the dashed arrows shown in the figure represent possible outsourcing paths of the traffic pieces from different Ingress Nodes. Different slices of the ingress network capacity C_k and the edge cloud computation capacity S_i are allocated to serve the different types of traffic based on the corresponding Service Level Agreements (SLAs), which, in this paper are focused on keeping latency under control. Thus, another objective of our model is to find the allocation of traffic to the edge clouds that allows us to minimize the total latency, which is expressed in terms of the latency at the ingress node, due to the limitations of the wireless network, plus the latency due to the traffic processing computation, plus the latency occurring in the communication links internal to the network system architecture.

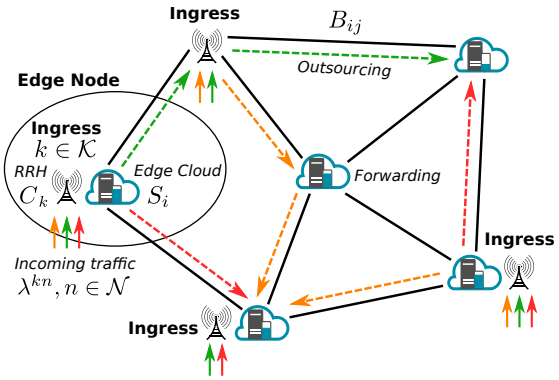
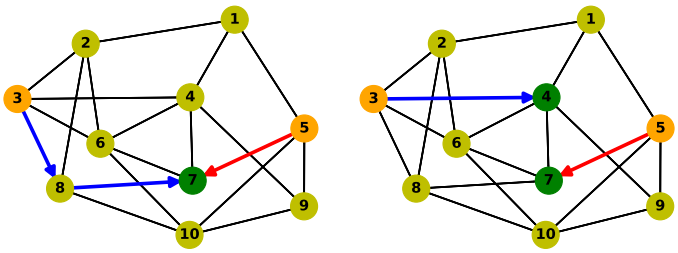


Fig. 1: Network system architecture.

We assume that the edge network is controlled by a *management component* which is in charge of achieving the optimal utilization of its resources, in terms of network and computation, still guaranteeing the SLA associated to each traffic type accepted by the network. This component mon-



(a) Minimizing both latency and computation costs

(b) With the same settings, but a change $\lambda^{n5,t2} = 40Gb/s$

Fig. 2: Toy example for a network with 10 nodes and 20 edges (average degree: 4.0).

itors the network by periodically computing the network capacity of each ingress node (through broadcast messages exchanged in the network) and the bandwidth of each link in the network topology. Moreover, it knows the maximum available computation capacity of all computation nodes. With these pieces of information as input, and knowing the SLA associated to each traffic type, the management component periodically solves an optimization problem that provides as output the identification of a proper network configuration and traffic allocation. In particular, it will identify: i) the amount of computational capacity to be assigned to each node so that, with the foreseen traffic, the node usage remains below a certain level of its capacity; ii) which node is taking care of which traffic type; and iii) the nodes through which each traffic type must be routed toward its destination.

For simplicity, the optimization problem is based on the assumption that the system is time-slotted, where time is divided into equal-length short slots (short periods where network parameters can be considered as fixed and traffic shows only small variations). We observe that our proposed heuristic (NESF) exhibits a short computing time so that it is feasible to run the problem periodically and to adjust the configuration of the system network based on the actual evolution of the traffic.

In the next section, we give an intuition of the solution applied by the management component in the case of a simple network, while in Section 4 we formalize the optimization problem and in Section 5 we present some heuristics that make the problem tractable in realistic cases.

3 Overview of Planning and Allocation

In this section we refer to a simple but still meaningful edge network and we show how the management component behaves in the presence of two types of traffic. In Section 4 we present in detail the optimization model that computes the allocation of computational and network resources as well as the optimal routing paths and we show how all values are computed. Here the goal is to provide the intuition beyond the proposed optimization approach.

The example we consider is shown in Figure 2 and consists of 10 nodes, two of which are ingress nodes (labeled as $n3$ and $n5$ in the figure and colored in orange), connected together with an average degree of 4. For simplicity, we assume that the bandwidth of all links is $B_l = 100Gb/s$, and the wireless network capacity of the two ingress nodes is, respectively, $C_{n3} = 50Gb/s$ and $C_{n5} = 60Gb/s$. Every node in the network has a computation capacity that can take one of the following values: $D_0 = 0Gb/s$ (i.e., no computation capacity is made available at the current time), $D_1 = 30Gb/s$, $D_2 = 40Gb/s$, and $D_3 = 50Gb/s$[1]. Given the above edge network, let us assume the management component estimates that node $n3$ will receive traffic of type $t1$ at rate $\lambda^{n3,t1} = 25Gb/s$ and type $t2$ at rate $\lambda^{n3,t2} = 20Gb/s$, while node $n5$ will receive the two types of traffic with rates $\lambda^{n5,t1} = 15Gb/s$ and $\lambda^{n5,t2} = 35Gb/s$, respectively. Finally, let us assume that the network operator has set an upper bound on the *power budget* to be used (i.e., the total amount of computational power) $P = 300Gb/s$ and has defined in its SLA a tolerable latency for the two types of traffic, respectively, to the following values: $\tau_{t1} = 1ms$ and $\tau_{t2} = 2ms$.

Under the above assumptions and constraints, the management component will solve the optimization problem, and will decide to offload part of the traffic from the two ingress nodes to an intermediate node as shown in Figure 2(a). More specifically, the management component will assign at ingress node $n3$ a wireless network capacity slice of $27Gb/s$ (out of the total $50Gb/s$) to $t1$ and of $23Gb/s$ to $t2$, while at ingress node $n5$ it will assign $22Gb/s$ (out of the total $60Gb/s$) to $t1$ and $38Gb/s$ to $t2$. Moreover, it will assign a computation capacity D_2 to nodes $n3$ and $n5$ and D_3 to $n7$, while it will switch off the computation capacity of the other nodes. This will lead to a total computation capacity of $130Gb/s$, which is well below the available computation capacity budget P. Given that $t1$ is the traffic type with the most demanding constraint in terms of latency, the management component decides to use the full D_2 capacity of $n3$ to process traffic $t1$ from $n3$. Applying the same strategy within node $n5$ would result in a waste of resources because the $t1$ traffic of $n5$ will take only $15Gb/s$ of the available computation capacity, and the remaining one will not be sufficient to handle the expected total amount of $t2$ traffic. Since moving the $t1$ traffic of one hop would still allow the system to fulfill the SLA, the decision will be then to configure the network to route such traffic to $n7$. The reason for choosing $n7$ is mainly because it is one of the nearest neighbors of both $n3$ and $n5$ (with 2 hops to $n3$ and 1 hop to $n5$) and, with its D_3 capacity, will be able to handle both $t2$ traffic from $n3$ and $t1$ traffic from $n5$. Specifically, the percentage of computation capacity of $n7$ allocated for $n3, t2$ and $n5, t1$ is 64% and 36%, respectively. $t2$ traffic from $n5$ is, instead, processed locally at $n5$ itself.

Let us now assume that the management component observes a change in the $\lambda^{n5,t2}$ traffic rate, which increases to $\lambda^{n5,t2} = 40Gb/s$. It will then run again the optimization algorithm that will output the configuration illustrated in Figure 2(b). The slicing of the wireless network capacity for ingress node $n3$ will not vary, while the total wireless capacity at ingress node $n5$ will be redistributed as follows: a slice of $42Gb/s$ will be assigned to $t2$ and, as a consequence, a slice of $18Gb/s$, smaller than before, to $t1$. Moreover, the

[1]Note that computation capacity is often expressed in cycles/s. As discussed in Section 6, for homogeneity with the other values, we have transformed it into Gb/s.

computation capacity to be allocated to each node will be recomputed. Capacity D_2 will be allocated to $n3$, which will process $t1$ locally, and D_1 will be allocated to the neighbor node $n4$, which will handle the $t2$ traffic from $n3$. Capacity D_3 will be allocated to $n5$ to process $t2$ locally and, finally, D_1 will be allocated to $n7$ to process $t1$ incoming from $n5$. Both ingress nodes will offload part of their traffic to the nearest 1-hop neighbor and the total computation capacity will be equal to $150Gb/s$.

Notice that, by manually analyzing the initial configuration of Figure 2(a), we may think that a better solution to the increase of $\lambda^{n5,t2}$ would be to simply increase the computation capacity of $n5$ to D_3 as in this way the network configuration will remain almost the same as before and the total computation capacity will be $140Gb/s$, smaller than the one of Figure 2(b). However, a more in-depth analysis shows that, even if this solution is certainly feasible, it is less optimal than the one of Figure 2(b) in terms of $t1$ total latency, which, as described in detail in Section 4, depends on both the wireless network latency and the outsourcing latency. The main reason for this increase in the latency is that traffic $t1$ from node $n5$ will suffer from a larger latency in the wireless ingress network due to a smaller allocated slice, and also from a relatively high latency due to the traffic computation on $n7$. According to the model we formalize in the next section, the total latency for $t1$ in this case is $0.72ms$, while, as it will be shown in Section 6.4, it is $0.47ms$ in the case of Figure 2(b), thanks to the fact that node $n5$ has the computation capacity of $n7$ entirely dedicated to the $t1$ traffic it introduces in the network.

4 Problem Formulation

In this section we provide the mathematical formulation of our *Joint Planning and Slicing of mobile Network and edge Computation resources* (JPSNC) model. Table 1 summarizes the notation used throughout this section. For brevity, we simplify expression $\forall n \in \mathcal{N}$ as $\forall n$, and apply the same rule to other set symbols like $\mathcal{E}, \mathcal{K}, \mathcal{L}$, etc. throughout the rest of this paper unless otherwise specified.

TABLE 1: Summary of used notations.

Parameters	Definition
$\mathcal{N}$	Set of traffic types
$\mathcal{E}$	Set of edge nodes in the edge networks
$\mathcal{K}$	Set of ingress nodes, where $\mathcal{K} \subseteq \mathcal{E}$
$\mathcal{L}$	Set of directed links in the networks
B_{ij}	Bandwidth of the link from node i to j, where $(i,j) \in \mathcal{L}$
C_k	Network capacity of ingress edge node $k \in \mathcal{K}$
D_a	Levels of computation capacities ($a \in \mathcal{A} = \{1, 2, 3 \ldots\}$)
P	Planning budget of computation capacity
λ^{kn}	User traffic rate of type n in ingress node k
τ_n	Tolerable delay for serving the total traffic of type n
κ_i	Cost of using one unit of computation capacity on node i
w	Weight to balance among total latency and operation cost
Variables	**Definition**
c^{kn}	Slice of the network capacity for traffic kn
b_i^{kn}	Whether traffic kn is processed on node i or not
α_i^{kn}	Percentage of traffic kn processed on node i
β_i^{kn}	Percentage of i's computation capacity sliced to traffic kn
δ_i^a	Decision for planning computation capacity on node i
$\mathcal{R}_i^{kn}$	Set of links for routing the traffic piece α_i^{kn} from k to i

The goal of our formulation is to minimize a weighted sum of the total latency and network operation cost for serving several types of user traffic under the constraints of users' maximum tolerable latency and network planning budget. This allows the network operator to fine tune its needs in terms of quality of service provided to its users and cost of the planned network. Different types of traffic, with heterogeneous requirements, need to be accommodated, and may enter the network from different ingress nodes.

In the following, we first focus on the network planning issue and its related cost, as well as on the traffic routing issue, and then detail all components that contribute to the overall latency experienced by users, which we capture in our model.

4.1 Network Planning and Routing

Network Planning: We assume that, in each edge node, some processing capacity can be made available, thus enabling MEC capabilities. This action will result in an operation cost that will increase at the increase of the amount of processing capacity. To model more closely real network scenarios, we assume that only a discrete set of capacity values can be chosen by the network operator and made available. Therefore, we adopt a piecewise-constant function S_i for the processing capacity of an edge node, in line with [8]. This is defined as:

$$S_i = \sum\nolimits_{a \in \mathcal{A}} \delta_i^a D_a, \ \forall i, \tag{1}$$

where D_a is a capacity level ($a \in \mathcal{A}$) and $\delta_i^a \in \{0, 1\}$ is a binary decision variable for capacity planning, satisfying the following constraint (only one level of capacity can be made available on a node, including zero, i.e., no processing capability):

$$\sum\nolimits_{a \in \mathcal{A}} \delta_i^a = 1 - \delta_i^0, \ \forall i, \tag{2}$$

where δ_i^0 is a binary variable that indicates whether node i has currently available some computation power or not. This constraint implies that S_i can be set as either 0 (no computation power) or exactly one capacity level, D_a.

To save on operation costs, in the case an edge node is not supposed to be exploited to process some traffic, then no processing capacity is made available on it. We introduce binary variable b_i^{kn} to indicate whether traffic kn is processed on node i (we will use the expression "traffic kn" in the following, for brevity, to indicate the user traffic of type n from ingress point k). Then the following constraint should be satisfied:

$$b_i^{kn} \leqslant 1 - \delta_i^0 \leqslant \sum_{k' \in \mathcal{K}} \sum_{n' \in \mathcal{N}} b_i^{k'n'}, \ \forall k, \forall n, \forall i, \tag{3}$$

We also consider a total planning budget, P, for the available computation capacity, introducing the following constraint:

$$\sum\nolimits_{i \in \mathcal{E}} S_i \leqslant P. \tag{4}$$

Then, the total operation cost can be expressed as:

$$J = \sum\nolimits_{i \in \mathcal{E}} \kappa_i S_i, \tag{5}$$

where κ_i is the cost of using one unit of computation capacity (in the example of Section 3 this will be $1Gb/s$) on node i.

Network Routing: We assume that each type of traffic can be split into multiple pieces only at its ingress node. Each piece can then be offloaded to another edge computing node independently of the other pieces, but it cannot be further split (we say that each piece is *unsplittable*).

The reason for using *unsplittable* routing in our optimization model is twofold: first of all, network slicing in the 5G architecture should be performed in an isolated manner for security and privacy reasons, especially for specific customer services [6, 9, 10]. Hence, considering unsplittable routing is, in practice, reasonable. Second, this choice is beneficial to reduce the complexity of our optimization problem since splitting the traffic across edge nodes could significantly increase the complexity without a strong justification, especially for the kind of user services mentioned above (with security and privacy requirements). In general, we consider that the user traffic or the virtual operator traffic passes through a predefined set of nodes along a given (unique) path, like a given chain of nodes providing services to the user/virtual provider.

Each link $l \in \mathcal{L}$ may carry different traffic pieces, α_i^{kn} (we denote by α_i^{kn} the percentage of traffic kn processed at node i, and with β_i^{kn} the percentage of computation capacity S_i sliced for traffic kn). Then, the traffic flow kn on l, f_l^{kn}, can be expressed as the sum of all pieces of traffic that pass through such link:

$$f_l^{kn} = \sum_{i \in \mathcal{E}:\ l \in \mathcal{R}_i^{kn}} \alpha_i^{kn},\ \forall k, \forall n, \forall l, \tag{6}$$

where $\mathcal{R}_i^{kn} \subset \mathcal{L}$ denotes a routing path (set of traversed links) for the traffic piece $\alpha_i^{kn}\lambda^{kn}$ from ingress k to node i. The following constraint ensures that the total traffic on each link does not exceed its capacity:

$$B_{ij} > \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} f_{ij}^{kn} \lambda^{kn},\ \forall (i,j) \in \mathcal{L}. \tag{7}$$

The traffic flow conservation constraint is enforced by the following constraint:

$$\sum_{j \in \mathcal{I}_i} f_{ji}^{kn} - \sum_{j \in \mathcal{O}_i} f_{ij}^{kn} = \begin{cases} \alpha_i^{kn} - 1, & \text{if } i = k, \\ \alpha_i^{kn}, & \text{otherwise,} \end{cases} \quad \forall k, \forall n, \forall i, \tag{8}$$

where $\mathcal{I}_i = \{j \in \mathcal{E} \mid (j,i) \in \mathcal{L}\}$ and $\mathcal{O}_i = \{j \in \mathcal{E} \mid (i,j) \in \mathcal{L}\}$ are the set of nodes connected by the incoming and outgoing links of node i, respectively. The fulfillment of this constraint guarantees *continuity* of the routing path. Moreover, the routing path $\mathcal{R}_i^{kn}$ should be *acyclic*.

To sum up, we consider at each ingress node aggregates of traffic, each corresponding to a type of traffic/service; an aggregate of type n at ingress node k has a total rate λ^{kn}. We split such aggregate (only) at ingress node k into several pieces $\{\alpha_i^{kn}\lambda^{kn}, i \in \mathcal{E}\}$, where α_i^{kn} represents the percentage of traffic kn processed at node i. We then determine for each piece $\alpha_i^{kn}\lambda^{kn}$ a single path $\mathcal{R}_i^{kn}$ between ingress node k and edge node i. Note that α_i^{kn} may be null for some edge nodes i and the selection of processing nodes depends, among other factors, on latency constraints specified in the next section since not all nodes are used to process a given traffic. In practice, since we deal with large aggregates, each single demand inside the aggregate follows a single path, (since it largely "fits" in the fraction of traffic that follows a single path).

4.2 Latency Components

The latency in each ingress edge node is modeled as the sum of the *wireless network latency* and the *outsourcing latency* which, in turn, is composed of the *processing latency* in some edge cloud and then *link latency* between edge clouds.

Wireless Network Latency: We model the transmission of traffic in each user ingress point as an $M|M|1$ processing queue. The *wireless network latency* for transmitting the user traffic of type n from ingress point k, denoted by t_W^{kn}, can therefore be expressed as:

$$t_W^{kn} = \frac{1}{c^{kn} - \lambda^{kn}},\ \forall k, \forall n, \tag{9}$$

where c^{kn} is the capacity of the network slice allocated for traffic kn in the ingress edge network (a decision variable in our model) and λ^{kn} is the traffic rate. The following constraints ensure that the capacity of all slices does not exceed the total capacity C_k of each ingress edge node, and c^{kn} is higher than the corresponding λ^{kn} value:

$$\sum\nolimits_{n \in \mathcal{N}} c^{kn} \leqslant C_k,\ \forall k, \tag{10}$$

$$\lambda^{kn} < c^{kn},\ \forall k, \forall n. \tag{11}$$

Processing Latency: We assume that each type of traffic can be segmented and processed on different edge clouds, and each edge cloud can slice its computation capacity to serve different types of traffic from different ingress nodes. As introduced before, we indicate with α_i^{kn} the percentage of traffic kn processed at node i, and with β_i^{kn} the percentage of computation capacity S_i sliced for traffic kn. The processing of user traffic is described by an $M|M|1$ model. Let $t_P^{kn,i}$ denote the *processing latency* of edge cloud i for traffic kn. Then, based on the computational capacity $\beta_i^{kn}S_i$ sliced for traffic kn, with an amount $\alpha_i^{kn}\lambda^{kn}$ to be served, $\forall k, \forall n, \forall i,\ t_P^{kn,i}$ is expressed as:

$$t_P^{kn,i} = \begin{cases} \frac{1}{\beta_i^{kn}S_i - \alpha_i^{kn}\lambda^{kn}}, & \text{if } \alpha_i^{kn} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

In the above equation, when traffic kn is not processed on edge cloud i, the corresponding value is 0; at the same time, no computation resource of i should be sliced to traffic kn (i.e., $\beta_i^{kn} = 0$). The corresponding constraint is written as:

$$\begin{cases} \alpha_i^{kn}\lambda^{kn} < \beta_i^{kn}S_i, & \text{if } \alpha_i^{kn} > 0, \\ \alpha_i^{kn} = \beta_i^{kn} = 0, & \text{otherwise.} \end{cases} \tag{13}$$

α_i^{kn} and β_i^{kn} also have to fulfill the following consistency constraints:

$$\sum\nolimits_{i \in \mathcal{E}} \alpha_i^{kn} = 1,\ \forall k, \forall n, \tag{14}$$

$$\sum\nolimits_{k \in \mathcal{K}} \sum\nolimits_{n \in \mathcal{N}} \beta_i^{kn} \leqslant 1,\ \forall i. \tag{15}$$

Link Latency: Let $t_L^{kn,i}$ denote the *link latency* for routing traffic kn to node i. In each ingress node, the incoming traffic is routed in a multi-path way, i.e., different types or pieces of the traffic may be dispatched to different nodes via different paths. $\forall k, \forall n, \forall i,\ t_L^{kn,i}$ is defined as:

$$t_L^{kn,i} = \begin{cases} \sum\limits_{l \in \mathcal{R}_i^{kn}} \frac{1}{B_l - \sum\limits_{k' \in \mathcal{K}} \sum\limits_{n' \in \mathcal{N}} f_l^{k'n'} \lambda^{k'n'}}, & \text{if } \alpha_i^{kn} > 0 \,\&\, i \neq k, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

Recall that $\mathcal{R}_i^{kn}$ is a routing path for the traffic piece $\alpha_i^{kn}\lambda^{kn}$ from ingress k to node i. The *link latency* is accounted for only if a certain traffic piece is processed on node i (i.e. $\alpha_i^{kn} > 0$) and $i \neq k$.

Total Latency: Now we can define the *outsourcing latency* for traffic kn, which depends on the longest serving time among edge clouds:

$$t_{PL}^{kn} = \max_{i \in \mathcal{E}} \{t_P^{kn,i} + t_L^{kn,i}\}, \ \forall k, \forall n. \tag{17}$$

The latency experienced by each type of traffic coming from the ingress nodes, can therefore be defined as $t_W^{kn} + t_{PL}^{kn}$, and also should respect the tolerable latency requirement:

$$t_W^{kn} + t_{PL}^{kn} \leqslant \tau_n, \ \forall k, \forall n. \tag{18}$$

For each traffic type n, we consider the maximum value among different ingress nodes with respect to the wireless network latency and outsourcing latency, i.e., $\max_{k \in \mathcal{K}}\{t_W^{kn} + t_{PL}^{kn}\}$. Then, we define the total latency as follows:

$$T = \sum_{n \in \mathcal{N}} \max_{k \in \mathcal{K}} \{t_W^{kn} + t_{PL}^{kn}\}. \tag{19}$$

The way we model latency and delay is aligned with other approaches in the literature. The work of Ma et al. [11] presents a system delay model which has the same components adopted in our paper; the communication delay in the wireless access is modeled as in our work (using an $M|M|1$-like expression). Moreover, this work also assumes that traffic is processed across a subset of computing nodes and the service time of edge hosts and cloud instances are exponentially distributed, hence the service processes of mobile edge and cloud can be modeled as $M|M|1$ queues in each time interval. The same assumption is made in [12]. In [13], the authors assume that both the congestion delay and the computation delay at each small-cell Base Station (by considering a Poisson arrival of the computation tasks) can be modeled as an $M|M|1$ queuing system; the work in [14] assumes that the baseband processing of each Virtual Machine (VM) on each User Equipment packets can be described as an $M|M|1$ processing queue, where the service time at the VM of each physical server follows an exponential distribution. Finally, the works [15–18] also adopt similar choices concerning the delay modeling.

4.3 Optimization Problem - JPSNC

Our goal in the *Joint Planning and Slicing of mobile Network and edge Computation resources* (JPSNC) problem is to minimize the total latency and the operation cost, under the constraints of maximum tolerable delay for each traffic type coming from ingress nodes and the total planning budget for making available processing-capable nodes:

$$\begin{aligned} \mathcal{P}0: \min_{\substack{c^{kn}, b_i^{kn}, \alpha_i^{kn}, \\ \beta_i^{kn}, \delta_i^a, \mathcal{R}_i^{kn}}} \quad & T + wJ, \\ \text{s.t.} \quad & (1) - (19), \end{aligned}$$

where $w \geq 0$ is a weight that permits to set the desired balance between the total latency and operation cost. Problem $\mathcal{P}0$ contains both nonlinear and indicator constraints, therefore, it is a mixed-integer nonlinear programming (MINLP) problem, which is hard to be solved directly [4], as discussed in Section 4.4.

We observe that we can give priority to one component of the objective function (latency T or operational cost J) with respect to the other by setting the weight w. This is obtained by setting w such that (if T is privileged) improving latency is preferred even if this increases the operational cost of the planned network at its maximum (a similar reasoning is applied if the cost J is privileged over T).

To this aim, we first compute the bounds for the values of T and J approximately as follows:

1) $\min(\kappa_i) \cdot \sum \lambda^{kn} \leqslant J \leqslant \max(\kappa_i) \cdot P$;
2) $|\mathcal{N}| \cdot (\frac{1}{\max(C_k)} + \frac{1}{\max(D_a)}) < T \leqslant \sum \tau_n$.

For the lower bound of $J = \sum(\kappa_i \cdot S_i) \geqslant \min(\kappa_i) \cdot \sum(S_i)$, we observe that the total computation power should cover the total traffic rate, to avoid infeasibility, hence we have: $\min(\kappa_i) \cdot \sum \lambda^{kn}$. For computing the bounds of T, we use its definition and the tolerable latency to get the upper bound, while for the lower bound, we use the definitions (wireless latency and computation latency, for link latency, we get 0 due to the lower bound) and let the denominators reach the maximum. The values of w that enforce the desired priority in the optimization process can therefore be computed as $w_L = \frac{T_{min}}{J_{max}}$ and $w_U = \frac{T_{max}}{J_{min}}$.

4.4 JPSNC Reformulation

Problem $\mathcal{P}0$ formulated in Section 4 cannot be solved directly and efficiently due to the following reasons:

- We aim at identifying the optimal routing (the routing path $\mathcal{R}_i^{kn}$ is a variable in our model, since many paths may exist from each ingress node k to a generic node i in the network); furthermore, we must ensure that such routing is acyclic and ensures continuity and unsplittability of traffic pieces.
- Variables $\mathcal{R}_i^{kn}$ and α_i^{kn} are reciprocally dependent: to find the optimal routing, the percentage of traffic processed at each node i should be known, and at the same time, to solve the optimal traffic allocation, the routing path should be known.
- The processing latency, defined in the previous sections, depends on three decision variables in our model and the corresponding formula (12) is (highly) nonlinear.
- $\mathcal{P}0$ contains indicator functions and constraints, e.g. (12), (13), (16), which cannot be directly and easily processed by most solvers.

To deal with the above issues, we propose an equivalent reformulation of $\mathcal{P}0$ (called Problem $\mathcal{P}1$), which can be solved very efficiently with the Branch and Bound method. Moreover, the reformulated problem can be further relaxed and, based on that, we propose in the next section an heuristic algorithm which can get near-optimal solutions in a shorter computing time. More specifically, in $\mathcal{P}1$, we first reformulate the processing latency and link latency constraints (viz., constraints (12) and (16)), and we deal, at the same time, with the computation planning problem. Then, we handle the difficulties related to variables $\mathcal{R}_i^{kn}$ and the corresponding routing constraints. Appendix A contains all details about the problem reformulation. Since some constraints are quadratic while the others are linear, $\mathcal{P}1$ is a mixed-integer quadratically constrained programming (MIQCP) problem, for which commercial and freely available solvers can be used, as we will illustrate in the numerical evaluation section.

5 HEURISTICS

Hereafter, we illustrate our proposed heuristic, named *Neighbor Exploration and Sequential Fixing* - NESF, which proceeds by exploring and utilizing the neighbors of each ingress node for hosting (a part of) the traffic along an *objective descent direction*, that is, by trying to minimize the objective function (which, we recall, is a weighted sum of the total latency and operation cost). During each step where we explore potential candidates for computation offloading, we partially fix the main binary decision variables in the reformulated problem $\mathcal{P}1$ and then solve the so-reduced problem by using the Branch and Bound method. Our exploring strategy provides excellent results, in practice, achieving near-optimal solution in many network scenarios, as we will illustrate in the Numerical Results Section.

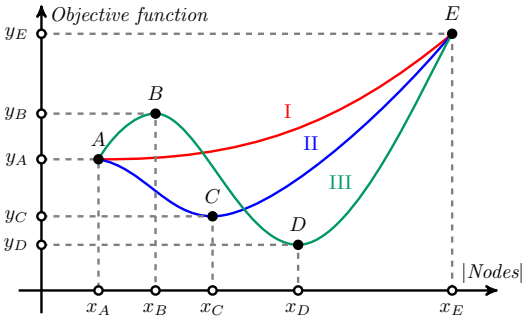


Fig. 3: Three typical variations of the objective function value versus the number of computing nodes made available.

The detailed exploring strategy is illustrated in Figure 3, which shows three typical variation paths of the objective function value versus the number of computing nodes made available in the network (note that these 3 trends are independent from each other, in the sense that either of them, or a combination of them, can be experienced in a given network instance). Point A represents the stage where a minimum required number of computing nodes (x_A) is opened to ensure the feasibility of the problem. For instance, if the ingress nodes can host all the traffic under all the constraints, $x_A = |\mathcal{K}|$. Point E indicates the maximum number of computing nodes that can be made available in the network; any point above x_E will violate the computation budget or tolerable latency constraints.

During the search phase of our heuristic, which is executed in Algorithms 1 and 3, detailed hereafter, we first try to obtain (or get as closer as possible to) point A and the corresponding objective value y_A. If A can not be found within the computation budget, the problem is infeasible. Otherwise, we continue to explore computation candidates from the h-hop neighbors of each ingress node, and allocate them to serve different types of traffic. The objective value is obtained by solving $\mathcal{P}1$ with new configurations of the decision variables. The change of the objective value may hence exhibit one of the three patterns (I, II and III) illustrated in Figure 3.

The objective value increases monotonically in path I. In path II, it first decreases to point C then increases to point E; finally, path III shows a more complex pattern which has one local maximum point B and one minimum point D.

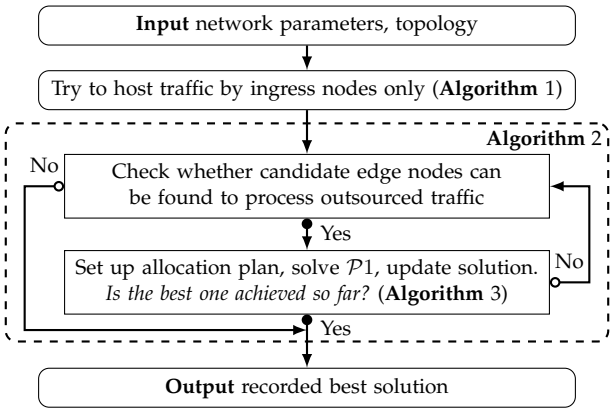


Fig. 4: Flowchart of our NESF heuristic.

In case I, the network system has just enough computation power to serve the traffic. Hence, adding more computation capacity to the system does not guarantee to decrease delay, while it will increase on the other hand computation costs. In case II, few ingress nodes in the system may support a relatively high traffic load. Equipping some of their neighbors with more computation capabilities (with total amount less than x_C) can still decrease the total system costs. After point C, the objective value shows a similar trend to case I. In case III, several ingress nodes may serve high traffic load. At the beginning, adding some computing nodes (with total amount less than x_B) may be not enough to decrease the delay costs to a certain degree, and this will also increase the total installation costs. After point B, the objective value varies like in case II and has a minimum at point D.

To summarize, our heuristic aims at reaching the minimum points A (I), C (II) and D (III) in Figure 3, and its flowchart is shown in Figure 4. The main idea behind Algorithm 1 is to check whether the ingress nodes can host all the traffic without activating additional MEC units, thus saving some computation cost. Algorithm 2 aims at searching the h-hop neighbors of each ingress node for making them process part of the traffic (the outsourced traffic), while Algorithm 3 aims at setting up the allocation plan for outsourced traffic and try to solve $\mathcal{P}1$ to obtain the best solution. The three proposed algorithms are described into detail in the following subsections. The definition of the new notation introduced in these algorithms is summarized for clarity in Table 2.

5.1 Attempt of serving traffic without additional MEC units

In Algorithm 1, the main idea is to check whether ingress nodes can host all the traffic, without using other MEC

TABLE 2: Notations used in the algorithms.

Notation	Definition
S_k^e	*Estimated* available computation of ingress node $k \in \mathcal{K}$
$\mathcal{K}^u$	Ingress nodes that cannot host all traffic ($S_k^e \leqslant 0$)
H	Maximum searching depth of our heuristic
$\mathcal{G}_k^h$	h-hop neighbors ($h \leqslant H$) of ingress node $k \in \mathcal{K}$
Q_k	Candidates for computing traffic from ingress node $k \in \mathcal{K}$
S_k^o	*Overall* computation of ingress node $k \in \mathcal{K}$
S_i^l	Maximum *left* computation of node $i \in \mathcal{E}$
$\mathcal{K}_i^b$	Ingress nodes who *booked* computation from node $i \in \mathcal{E}$
d_{ik}	Count of hops from node i to ingress node $k \in \mathcal{K}$
$O_{\mathcal{P}}$	Objective function value of problem $\mathcal{P}$

Algorithm 1 *Attempt of serving traffic with ingress nodes only*

1: $S_k^e = D_m - \sum_{n\in\mathcal{N}} \lambda^{kn},\ \forall k \in \mathcal{K}$;
2: $\mathcal{K}^u = \{k \in \mathcal{K} \mid S_k^e \leqslant 0\}$;
3: Compute k's h-hop neighbors $\mathcal{G}_k^h,\ h \leqslant H,\ \forall k \in \mathcal{K}$;
4: $Q_k = \{k\},\ \forall k \in \mathcal{K},\ O_t = -1$;
5: **for** $k \in \mathcal{K}^u$ **do**
6: $\quad \mathcal{X} = \{k' \in [(\mathcal{K} - \mathcal{K}^u) \cap (\bigcup_h^H \mathcal{G}_k^h)] \mid S_{k'}^e + S_k^e > 0\}$;
7: $\quad Q_k = Q_k \cup \mathcal{X}$, rank Q_k by increasing hop count to k;
8: Rank $\mathcal{N}$ as $\mathcal{N}_k$ by descending $(\lambda^{kn}, \tau_n), \forall k \in \mathcal{K}$;
9: **if** $\mathcal{K}^u = \varnothing$ **or** $\bigwedge_{k\in\mathcal{K}^u}(|Q_k| > 1)$ **then**
10: $\quad$ Allocate Q_k to $\mathcal{N}_k$ in order and repeatedly, $\forall k \in \mathcal{K}$;
11: $\quad$ Solve $\mathcal{P}1$ by **B&B** to obtain obj. fct. value $O_{\mathcal{P}1}$;
12: $\quad$ **if** $O_{\mathcal{P}1} > 0$ **then** $O_t = O_{\mathcal{P}1}$;

units in order to save both computation cost and latency. To this end, we first individuate the subset of ingress nodes (denoted as $\mathcal{K}^u$) that cannot host all the traffic that enters the network through them. This is done by checking if S_k^e $(= D_m - \sum_{n\in\mathcal{N}} \lambda^{kn}) \leqslant 0$ (lines 1-2), that is, if some computing capacity is still available or not at ingress nodes (recall that D_m is the maximum computation capacity that can be made available). Then, if $\mathcal{K}^u \neq \varnothing, \forall k \in \mathcal{K}^u$, we try to find the set of its neighbor ingress nodes $k' \in [(\mathcal{K} - \mathcal{K}^u) \cap (\bigcup_h^H \mathcal{G}_k^h)]$ that can cover S_k^e (i.e., $S_{k'}^e + S_k^e > 0$), where $\mathcal{G}_k^h \subset \mathcal{E}$ is the set of node k's h-hop neighbor nodes ($h = 1, \ldots, H$). If found, they are stored as candidates in a list, Q_k, ordered with increasing distance (hop count) from k (lines 3-7). If $\mathcal{K}^u = \varnothing$ or sufficient nodes in Q_k have been found to process the extra traffic from $\mathcal{K}^u$ (line 9), then $\forall k \in \mathcal{K}^u$, the corresponding traffic is allocated to nodes in Q_k starting from the top (choosing the closest ones) and repeatedly (covering all the traffic types), beginning with less latency to more latency-tolerant traffic.

This is implemented by setting the corresponding variables b_i^{kn}, δ_i^a and $\gamma_l^{kn,i}$ in $\mathcal{P}1$ to save the costs and also accelerate the algorithm. Finally, $\mathcal{P}1$ with the fixed variables is solved by using *Branch and Bound* method to obtain the solution (lines 10-11). If $\mathcal{P}1$ is feasible with these settings, the objective value $O_{\mathcal{P}1}$ is stored to be used in the next searching and resource allocation phases of Algorithm 3.

5.2 Neighbor search for computation candidates

This section describes Algorithm 2, upon which Algorithm 3 is based to provide the final solution. Algorithm 2 proceeds as follows. We first assign a rank (or a priority value) to each ingress node taking into account the amount of incoming traffic and the computation capacity. Then, we handle the outsourced traffic offloading task (i.e., choose the best subset of computational nodes) starting from the ingress node with the highest priority.

In more detail, set $\mathcal{K}^s$ is set $\mathcal{K}$ sorted by the ascending value of the tuple $(S_k^e, -\lambda^{kn})$, i.e., the ingress node with the lowest estimated available (left) computation S_k^e and the higher amount of traffic of type n has the highest rank/priority in our Algorithm 2, where n represents the traffic type having the maximum tolerable latency (lines 1-2). The process of determining the best subset of computation nodes for processing the outsourced traffic of each ingress node is executed *hop-by-hop*, starting with ingress node $\hat{k} = \mathcal{K}^s(0)$, until any one of the following three conditions is satisfied:

Algorithm 2 *Priority searching of computation candidates*

1: Rank ingress nodes as $\mathcal{K}^s$ by ascending $(S_k^e, -\lambda^{kn})$;
2: $\hat{k} = \mathcal{K}^s(0), h_k = 1, S_k^o = S_k^e\,(\forall k \in \mathcal{K}), \mathcal{K}_i^b = \varnothing\,(\forall i \in \mathcal{E})$;
3: **while** $|\bigcup_{k\in\mathcal{K}} Q_k| < \lfloor \frac{P}{min(D_a)} \rfloor$ **and** $\mathcal{K}^s \neq \varnothing$ **do**
4: $\quad \mathcal{B} = \varnothing$;
5: $\quad$ **for** $i \in (\mathcal{G}_{\hat{k}}^{h_{\hat{k}}} - [\mathcal{K} \cup Q_{\hat{k}}])$ **do**
6: $\quad\quad S_i^l = D_m + \sum_{k\in\mathcal{K}_i^b} S_k^e$
7: $\quad\quad$ **if** $S_i^l + S_{\hat{k}}^e > 0$ **then** $\mathcal{B} = \mathcal{B} \cup \{i\}$;
8: $\quad$ **if** $\mathcal{B} = \varnothing$ **then**
9: $\quad\quad h_{\hat{k}}{+}{+}$, update $\mathcal{K}^s, \hat{k}$ when $h_{\hat{k}} > H$ and **continue**;
10: $\quad$ Rank $\mathcal{B}$ by descending $(S_i^l, -d_{ik} : k \in \mathcal{K}^s)$, $\hat{\imath} = \mathcal{B}(0)$;
11: $\quad Q_{\hat{k}} = Q_{\hat{k}} \cup \{\hat{\imath}\},\ \mathcal{K}_{\hat{\imath}}^b = \mathcal{K}_{\hat{\imath}}^b \cup \{\hat{k}\},\ S^b = D_m$;
12: $\quad$ **for** $k \in \mathcal{K}^s \backslash \{\hat{k}\}$, **if** $(\hat{\imath} \in \bigcup_h^H \mathcal{G}_k^h)\ \&\ (S^b > \overline{\lambda^k})$ **do**
13: $\quad\quad Q_k = Q_k \cup \{\hat{\imath}\},\ \mathcal{K}_{\hat{\imath}}^b = \mathcal{K}_{\hat{\imath}}^b \cup \{k\},\ S^b = S^b - \overline{\lambda^k}$;
14: $\quad S_k^o = S_k^o + (D_m + \sum_{k'\in\mathcal{K}_{\hat{\imath}}^b \cap \mathcal{K}^u - \{k\}} S_{k'}^e),\ \forall k \in \mathcal{K}_{\hat{\imath}}^b$;
15: $\quad \hat{k} = \mathrm{argmin}_{k\in\mathcal{K}^s} S_k^o$;
16: $\quad$ **if** $S_{\hat{k}}^o \leqslant 0$ **then continue**; **else** $skip := (S_{\hat{k}}^o \leqslant r D_m)$;
17: $\quad$ Run (**Algorithm** 3) to obtain O_t;
18: **Return** O_t;

(1) the number of computation nodes opened for processing traffic exceeds the maximum budget $\lfloor \frac{P}{min(D_a)} \rfloor$, or
(2) all ingress nodes are completely scanned (line 3), or
(3) the algorithm could not improve further the solution (Algorithm 3, lines 8, 10).

In the searching phase, we first try to identify the set of temporary candidate computation nodes $\mathcal{B}$ for ingress $\hat{k}$ $(\mathcal{B} \subseteq (\mathcal{G}_{\hat{k}}^{h_{\hat{k}}} - [\mathcal{K} \cup Q_{\hat{k}}]))$, by checking if the maximum available computation capacity of $i \in \mathcal{B}$, S_i^l could help $\hat{k}$ to cover $S_{\hat{k}}^e$ (lines 4-7). S_i^l is computed as the difference between i's maximum installable computation capacity D_m and the total computation booked from i by ingress nodes in $\mathcal{K}_i^b \subseteq \mathcal{K}$, i.e., $\sum_{k\in\mathcal{K}_i^b} S_k^e$, where $\mathcal{K}_i^b$ is the set of ingress nodes that booked computation from node i. If $\mathcal{B} = \varnothing$, we increase the number of hops $h_{\hat{k}}$ for ingress $\hat{k}$. If not (we are done with $\hat{k}$), we move to the next ingress node in the set $\mathcal{K}^s$ (lines 8-9).

At this point we rank $\mathcal{B}$ by descending values of tuple $(S_i^l, -d_{ik} : k \in \mathcal{K}^s)$, where d_{ik} is the count of hops from node i to ingress node $k \in \mathcal{K}^s$. The first computation node $\hat{\imath}$ is selected as the one to compute the traffic of $\hat{k}$, and $\hat{k}$ is added into the corresponding set $\mathcal{K}_{\hat{\imath}}^b$. To make full use of computation node $\hat{\imath}$, we further spread it to help other ingress nodes $\mathcal{K}^s \backslash \{\hat{k}\}$, if $\hat{\imath}$ is their neighbor within H hops and has sufficient computation budget (lines 10-13). Then, given such computation node $\hat{\imath}$ and for each ingress node k, we update the value of the overall computation, S_k^o, due to the full use of computation nodes $\hat{\imath}$ (line 14). Hence, ingress k with the minimum support S_k^o will be chosen as the next searching target and Algorithm 2 continues as follows.

The next searching target $\hat{k}$ is set to $k \in \mathcal{K}^s$ with the minimum S_k^o value (lines 15-16). If $S_{\hat{k}}^o \leqslant 0$, this means that the current computation configuration could not host all the traffic; hence, the algorithm will go back to the *while* loop and *continue* to the next searching. Otherwise, we set a flag $skip := (S_{\hat{k}}^o \leqslant r D_m)$ where r is set to a small value (i.e.,

Algorithm 3 *Allocating resources and obtaining the solution*

1: Relax $b_i^{kn}, \delta_i^a, \gamma_l^{kn,i}$ to continuous ones ($\mathcal{P}1 \rightarrow \tilde{\mathcal{P}}1$);
2: Allocate Q_k to $\mathcal{N}_k$ *partially* and solve $\tilde{\mathcal{P}}1$ to obtain $\tilde{b}_i^{kn}$;
3: **if** $O_{\tilde{\mathcal{P}}1} > 0$ **then**
4: Rank candidates as Q_k^s by descending $\sum_{n\in\mathcal{N}} \tilde{b}_i^{kn}$;
5: Revert to the original problem $\mathcal{P}1$;
6: **if** $O_t > 0$ **then** set O_t as $\mathcal{P}1$'s *upper bound*;
7: Allocate Q_k^s to $\mathcal{N}_k$ and solve $\mathcal{P}1$;
8: **if** $0 < O_t \& (O_t < O_{\mathcal{P}1} || O_{\mathcal{P}1} < 0) \& \overline{skip}$ **then break**;
9: **if** $0 < O_{\mathcal{P}1} \& (O_{\mathcal{P}1} < O_t || O_t < 0)$ **then** $O_t = O_{\mathcal{P}1}$;
10: **else if** $O_t > 0$ & $\overline{skip}$ **then break**;

0.1). If $skip$ is *true*, it indicates that $\hat{k}$ has a high traffic load, and this may cause the processing latency to increase. This flag is used in Algorithm 3. In fact, this step implements the strategy of skipping point B to avoid the local minimum (point A) in path III shown in Figure 3. Finally, based on Q_k, we run Algorithm 3 to obtain the objective value O_t and the corresponding solution.

5.3 Resource Allocation and Final Solution

In Algorithm 3, we first relax problem $\mathcal{P}1$ to $\tilde{\mathcal{P}}1$, replacing binary variables b_i^{kn}, δ_i^a and $\gamma_l^{kn,i}$ with continuous ones. Given the set Q_k (by Algorithm 2) of candidate computation nodes for processing the outsourced traffic of ingress node k, the goal is to allocate node k's different traffic types to the computation nodes in Q_k starting with the traffic with the most stringent constraint in terms of latency. Unused computation nodes are turned off. These two steps (lines 1-2) provide a partial guiding information and also an acceleration for solving the relaxed problem, thus obtaining quite fast the relaxed optimal values of $\tilde{b}_i^{kn}$.

If $\tilde{\mathcal{P}}1$ is infeasible ($O_{\tilde{\mathcal{P}}1} < 0$), we check whether both the previous best solution exists ($O_t > 0$) and the algorithm does not $skip$. If yes, the searching process breaks and returns O_t (line 10). Otherwise, the algorithm will continue searching to avoid getting stuck in a local optimum point in path III (see Figure 3), according to the following.

Hence, if $\tilde{\mathcal{P}}1$ is feasible (line 3), the obtained $\tilde{b}_i^{kn}$ value can be regarded as the probability of processing traffic kn at node i. Based on this, for each ingress k, we rank the candidates in descending order of the probabilities $\sum_{n\in\mathcal{N}} \tilde{b}_i^{kn}$. Then we revert to the original problem $\mathcal{P}1$, set the upper bound for $\mathcal{P}1$ if possible, allocate the candidates to host all types of traffic in order and repeatedly for each ingress node, and also turn off the unused nodes (lines 5-7). By solving $\mathcal{P}1$, we obtain the current solution and compare it with the previous best one (O_t). If the solution gets worse, the whole searching process breaks out and returns the recorded best result (line 8). Otherwise (if the solution is improving), the current solution is updated as the best one and the searching process continues.

5.4 Summary and Acceleration Technique

Essentially, the proposed heuristic described in the above subsections exploits the $\mathcal{P}1$ formulation limiting the search space only to the nodes that are within a limited number of hops $h < H$ from the ingress nodes. We expect this is a realistic assumption based on the consideration that the main purpose of edge networks is to keep the traffic as close as possible to the ingress nodes and, therefore, to the users. Thanks to this approach, we are able to make the $\mathcal{P}1$ problem more tractable and solvable in a short time even in the case of complex edge networks (see Section 6).

We can further improve the solution time by eliminating from the problem formulation all unneeded variables. In particular, we modify $\mathcal{P}1$ by adding a scope k (where k is the ingress node) to $\mathcal{E}$ and $\mathcal{L}$. $\mathcal{E}_k \subseteq \mathcal{E}$ represents the set of h-hop neighbor nodes ($h \leqslant H$) of k and $\mathcal{L}_k \subseteq \mathcal{L}$ the set of links inside this neighborhood. This way, the solver will be able to skip all variables outside the considered k scope, thus reducing the time needed to load, store, analyze and prune the problem. Such modification does not change the result produced by the heuristic but it results in a consistent improvement (up to 1 order of magnitude) in the computing time needed to obtain the solution in our numerical analysis.

6 Evaluation

The goal of this evaluation is to show that: i) our $\mathcal{P}1$ model offers an appropriate solution to the edge network optimization problem we have discussed in this paper, ii) our *NESF* heuristic computes a solution which is aligned with the optimal one, and iii) when compared with two benchmark heuristics, *Greedy* and *Greedy-Fair*, *NESF* offers better results within similar ranges of computing time.

Consistently, the rest of this section is organized as follows: Section 6.1 describes the heuristics we have compared with; Section 6.2 presents the network topologies we have considered in the experiments; Section 6.3 describes the setup for our experiments; Section 6.4 discusses about optimal solution and the results obtained by the heuristics in the small network scenario presented in Section 3; Section 6.5 analyzes the results achieved by the heuristics when the network parameters vary; Finally, Section 6.6 discusses about the computing time needed to find a solution.

6.1 Benchmark Heuristics

We propose two benchmark heuristics, based on a greedy approach, which can be naturally devised in our context:

Greedy: With this approach, each ingress node uses its neighbor nodes computation facilities to guarantee a low overall latency for its incoming traffic. Hence, each ingress node first tries to locally process all incoming traffic. If its computation capacity is sufficient, a feasible solution is obtained; otherwise, the extra traffic is split and outsourced to its 1-hop neighbors, and so on, until it is completely processed (if possible).

Greedy-Fair: It is a variant of *Greedy* which performs a sort of *"fair" traffic offloading* on neighbor nodes. More specifically, it proceeds as follows: 1) compute the maximum number of available computing nodes, based on the power budget and the average computation capacity of a node; 2) divide such maximum number (budget) into $|\mathcal{K}|$ parts according to the ratio of the total traffic rate among ingress nodes, and choose for each ingress node the corresponding number of computing nodes from its nearest h-hop neighbors. Each ingress node spreads its load on its neighbors proportionally to the corresponding distance ($\frac{1}{hop+1}$), for example, if the load is outsourced to two *1-hop* neighbors, the ratio is $(1 : \frac{1}{2} : \frac{1}{2}) = (0.5 : 0.25 : 0.25)$.

6.2 Network Topologies

We experimented with our optimization approach using multiple network topologies.

6.2.1 Random graphs

We exploited Erdös-Rényi random graphs [19] by specifying the number of nodes and edges. As the original Erdös-Rényi algorithm may produce disconnected random graphs with isolated nodes and components, to generate a connected network graph, we patched it with a simple strategy that connects isolated nodes to randomly sampled nodes (up to 10 nodes) in the graph. We generated several kinds of topologies with different numbers of nodes and edges, shown in Figure 5, that span from a quasi-tree shape topology (Figure 5(c)) to a more general, highly connected one with 100 nodes and 150 edges (Figure 5(f)). The structural information for all topologies is shown in Table 3. All topology datasets are publicly available in our repository[2]. These topologies can be considered representative of various edge network configurations where multiple edge nodes are distributed in various ways over the territory. Due to space constraints, in the following we present and discuss the results obtained for a representative topology, i.e., the one in Figure 5(e), as well as those for the small topology of Figure 2, used to compare our proposed heuristics to the optimal solution. The full set of results is available online[3].

TABLE 3: Structural information of the topologies used in the experiments.

Topology	#Node	#Edge	#Ingress	Degree (Min, Max, Avg)	Diameter
10N20E	10	20	2	(3.0, 5.0, 4.0)	3
20N30E	20	30	3	(1.0, 5.0, 3.0)	6
40N60E	40	60	3	(1.0, 7.0, 3.0)	8
50N50E	50	50	3	(1.0, 4.0, 2.0)	15
60N90E	60	90	3	(1.0, 6.0, 3.0)	7
80N120E	80	120	3	(1.0, 6.0, 3.0)	9
100N150E	100	150	3	(1.0, 7.0, 3.0)	9
Città Studi	30	35	6	(1.0, 6.0, 2.3)	10

6.2.2 A real network scenario

We further considered a real network scenario, with the actual deployment of Base Stations (BSs) collected from the open database OpenCellID[4]. Specifically, we considered the "Città Studi" area around Politecnico di Milano and selected one mobile operator (Vodafone) with 133 LTE cells falling in such area (see Figure 6(a)). The BSs deployment shows where the BSs are located but it does not show their interconnection topology nor where the edge clouds are deployed. The reader should note that it is not easy to have access to such piece of information as it is both sensitive for the mobile operator and in continuous evolution. To the best of our knowledge, there is no publicly available true BSs interconnection topology, and for this reason, we decided to infer one as described below. We performed a clustering on the LTE cells, as illustrated in Figure 6(b), obtaining 30 clusters. Finally, we generated the network topology which, as in real mobile scenarios, has a fat tree-like shape with nodes connecting to other nodes. More specifically, starting from the cluster centroids, we connected any two nodes if the distance is lower than a given threshold (800 meters). By doing so, note that some "leaf" nodes become connected to more than one *aggregation node* – i.e., a node that is reached by multiple other nodes – to increase redundancy and hence reliability of the final topology, as it happens in real networks; finally, we generated the Minimum Spanning Tree of the geometric graph weighted by the distance and cluster size, while preserving redundant links. The resulting topology is illustrated in Figure 6(c); the average node degree resulting from the above procedure is 2.33 and edge clouds can be installed in all nodes (as suggested by 5G specifications). The structural information for this topology is shown in the last row of Table 3.

[2]https://github.com/bnxng/Topo4EdgePlanning

[3]http://xiang.faculty.polimi.it/files/SupplementaryResults.pdf

[4]https://www.opencellid.org

TABLE 4: Parameters setting - Initial (reference) values (for the case of high traffic load with low tolerable latency)

Parameter		Initial value	
Link bandwidth B_l	(Gb/s)	100	$(l \in \mathcal{L})$
Network capacity C_k	(Gb/s)	60, 50, 40	$(k \in \mathcal{K})$
Computation level D_a	(Gb/s)	30, 40, 50	$(a \in \mathcal{A})$
Computation budget P	(Gb/s)	300	
Traffic rate λ^{kn}	(Gb/s)	$\begin{bmatrix} 5 & 20 & 7 & 9 & 15 \\ 16 & 4 & 12 & 8 & 6 \\ 7 & 9 & 3 & 12 & 5 \end{bmatrix}$	$(\mathcal{K} \times \mathcal{N})$
Tolerable latency τ_n	(ms)	1, 1.5, 2, 3, 3.5	$(n \in \mathcal{N})$
Weights κ_i, w		0.1, 0.1	$(i \in \mathcal{E})$

6.3 Experimental Setup

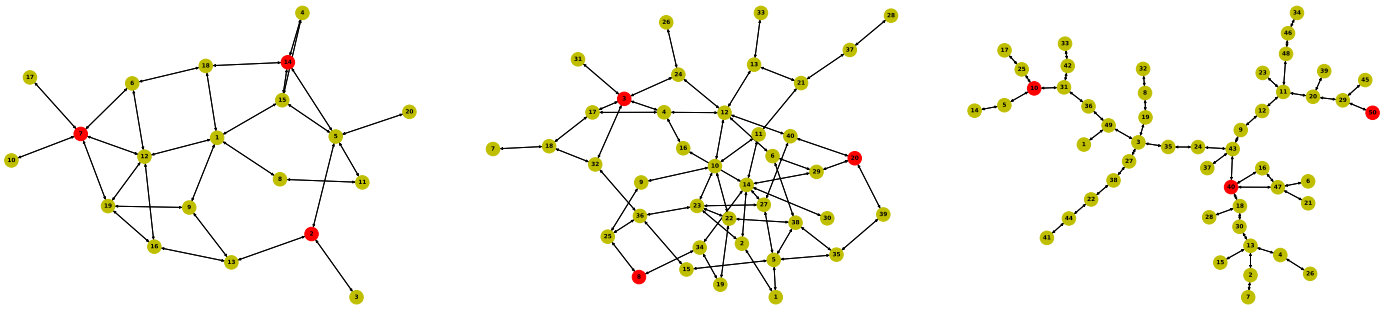
We implement our model and heuristics using SCIP (Solving Constraint Integer Programs)[5], an open-source framework that solves constraint integer programming problems. All numerical results presented in this section have been obtained on a server equipped with an Intel(R) Xeon(R) E5-2640 v4 CPU @ 2.40GHz and 126 Gbytes of RAM. The parameters of SCIP in our experiments are set to their default values.

The illustrated results are obtained by averaging over 50 instances with random traffic rates λ^{kn} following a Gaussian distribution $N(\mu, \sigma^2)$, where μ is the value of λ^{kn} shown in Table 4 and $\sigma = 0.1$ (we recall that the optimization problem is solved under the assumption that the traffic shows only little random variations during the time slot under observation. For this reason, the choice of a Gaussian distribution is appropriate). We computed 95% narrow confidence intervals, as shown in the following figures.

In Table 4 we provide a summary of the reference values we define for each parameter for the experiments with the random topologies. Such values are representative of a scenario with a high traffic load and low tolerable latency relative to the limited communication and computation resources. Referring to the computation capacity levels and budget in Table 4, it is worth noticing that unit "cycles/s" is often used for these metrics; for simplicity we transform it into "Gb/s" by using the factor "8bit/1900cycles", which assumes that processing 1 byte of data needs 1900 CPU cycles in a BBU pool [17].

The number of traffic types is set to five. Each traffic type can be dedicated to a specific application case (e.g.,

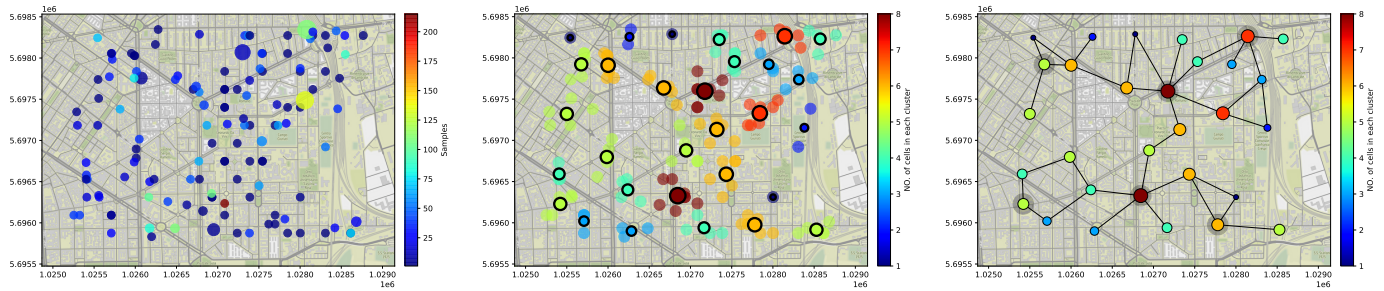
[5]http://scip.zib.de



(a) 20 nodes 30 edges (b) 40 nodes 60 edges (c) 50 nodes 50 edges

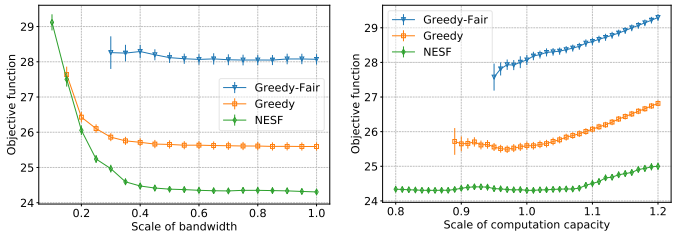
(d) 60 nodes 90 edges (e) 80 nodes 120 edges (f) 100 nodes 150 edges

Fig. 5: Network topologies. Ingress nodes for each graph are colored in red.



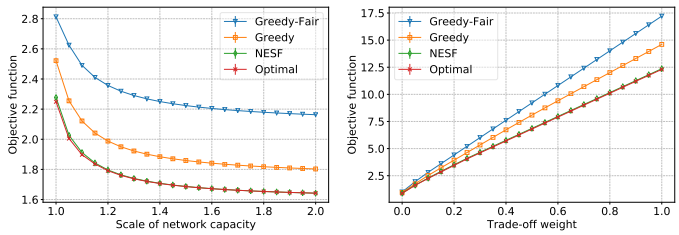
(a) Vodafone LTE cells (b) Cell clusters (c) Topology on clusters

Fig. 6: Città Studi topology with 30 nodes, 35 edges and 6 ingress nodes (marked with gray shadow).



(a) Link bandwidth (B_l) (b) Computation capacity (D_3)

Fig. 7: Selected numerical results for Città Studi topology.



(a) Network capacity (C_k) (b) Trade-off weight (w)

Fig. 8: Comparison with the optimum varying two selected parameters (C_k and w) in the example network scenario 10N20E of Figure 2.

video transmission for entertainment, real-time signaling, virtual reality games, audio). Our traffic rates result from the aggregation of traffic generated by multiple users connected at a certain ingress nodes. We select rate values that can be typical in a 5G usage scenario and that almost saturate the wireless network capacity at the ingress nodes that we assume to vary from 40 to 60 Gb/s. The tolerable latency for each traffic type aims at challenging the approach with quite demanding requirements ranging from 1 to 3.5 ms. More specifically, the values of traffic rate λ^{kn} and tolerable latency τ_n are designed to cover several different scenarios, i.e., *mice*, *normal* and *elephant* traffic load under *strict*, *normal* and *loose* latency requirements. For simplicity, in this paper we fix the number of ingress nodes to three. An in-depth analysis of the impact of the number of ingress nodes on the performance of the optimization algorithm is the subject of our future research. To make the problem solution manageable, we assume to adopt links of the same bandwidth (100 Gb/s) that are representative of current fiber connections. As in the example of Section 3, we assume three possible levels for the computation capacity (30, 40 and 50 Gb/s), under the assumption that, as it happens in typical cloud IaaS, users see a predefined computation service offer. The maximum computation budget is set to 300 Gb/s,

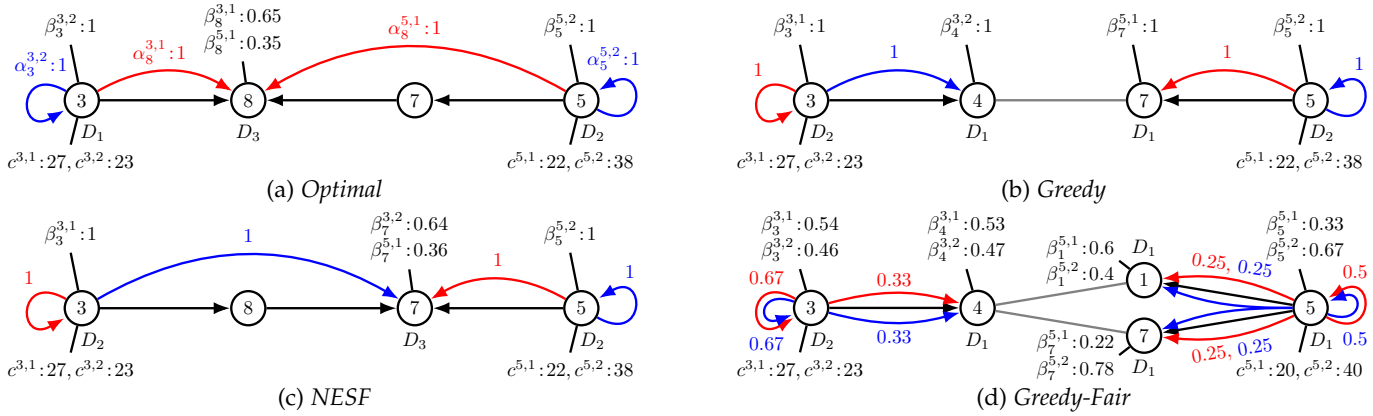


Fig. 9: Comparison of the solutions achieved by the heuristics and the optimum for the 10N20E topology.

which is a relatively low value considering the traffic rates we use in the experiments and the number of available nodes in the considered topologies. Finally, by assigning the same values to weights κ_i, w, we make sure that the two components of the optimization problem, the total latency and the operation cost, have the same importance in the identification of the solution.

In the network scenario of Section 6.2.2, we set the network capacity of each edge (i, j) proportionally to the size of nodes/clusters to make it scale by a factor K (set according to the specific parameters of our network scenario to 12.5, more precisely using expression $12.5 \cdot \max_{n\in\{i,j\}}\{\#\{\text{Node}\}_n\}$) so that, as in real mobile access networks, it can accommodate aggregate traffic coming from edge/leaf nodes to aggregation nodes. Finally, we select 6 ingress nodes (marked by gray shadow in Figure 6(c)), and the traffic rates in Table 4 are correspondingly duplicated from 3 to 6, while the planning budget is increased to $P = 600Gb/s$ for this scenario.

On such topology, we run the numerical experiments, and the results show very similar trends as those illustrated in Figure 10. In Figure 7 we chose a subset of the results (the objective function value of our optimization model) obtained by scaling the link bandwidth B_l and the computation capacity D_3 (those for the network capacity C_k are shown in Fig. 12(c)).

6.4 Analysis of the optimization results for a small network

We first compare the results obtained by our proposed heuristic, *NESF*, against the optimum obtained solving model $\mathcal{P}1$ in the simple topology illustrated in Figure 2, Section 3. Note that the original model could be solved only in such a small network scenarios due to a very high computing time. In Figure 8 we show the variation of the objective function (the sum of total latency and operation cost) with respect to two parameters, the network capacity C_k and the weight w in the objective function. In these cases, it can be observed that *NESF* obtains near-optimal solutions, practically overlapping with the optimum curve, for the whole range of the parameters, while both *Greedy* and *Greedy Fair* perform worse. The results achieved when the other parameters vary show the same trend. For the sake of space, we do not show them, but they are reported in the supplementary results available here[3].

Figure 9 shows the configuration of nodes and routing paths for the network (10N20E) with the parameter values defined in Section 3. Each sub-figure refers to one of the four considered solutions. Here we highlight the ingress nodes (i.e., 3 and 5) and the other nodes which offer computation capacity or support traffic routing. The remaining nodes are not shown for the sake of clarity. The black arrows represent the enabled routing paths. The traffic flow allocation of each solution is marked in red for traffic type 1 and blue for type 2, respectively. The values of all relevant decision variables (see Section 4) are shown as well.

Comparing Figures 9(a) and 9(c), we notice that both *Optimal* and *NESF* enable the computation capacity on the ingress nodes and an intermediate node, with one type of traffic kept in the ingress nodes and the other offloaded to the intermediate. The obvious differences between *Optimal* and *NESF* include: i) planning of the computation capacity on ingress node 3 (i.e., D_1 by *Optimal* while D_2 by *NESF*), and ii) the intermediate node selected and the consequent routing paths. However, the obtained objective function values (trade-off between the total latency and operation cost) by *Optimal* and *NESF* are respectively 2.25 and 2.28, and very close to each other. To further check the reasons behind, we found that the latencies for the traffic of type 1 and 2 are, respectively, $0.49ms$ and $0.55ms$ for *Optimal*, while $0.50ms$ and $0.47ms$ for *NESF*. Since in this case *NESF* can acquire less total latency at the expense of a little bit higher computation cost, compared with *Optimal*, their corresponding objective function values are close. Note that the computing time needed to obtain the optimal solution is around 10 hours (35724 seconds) while *NESF* is able to compute the approximate solution in only about 1 second.

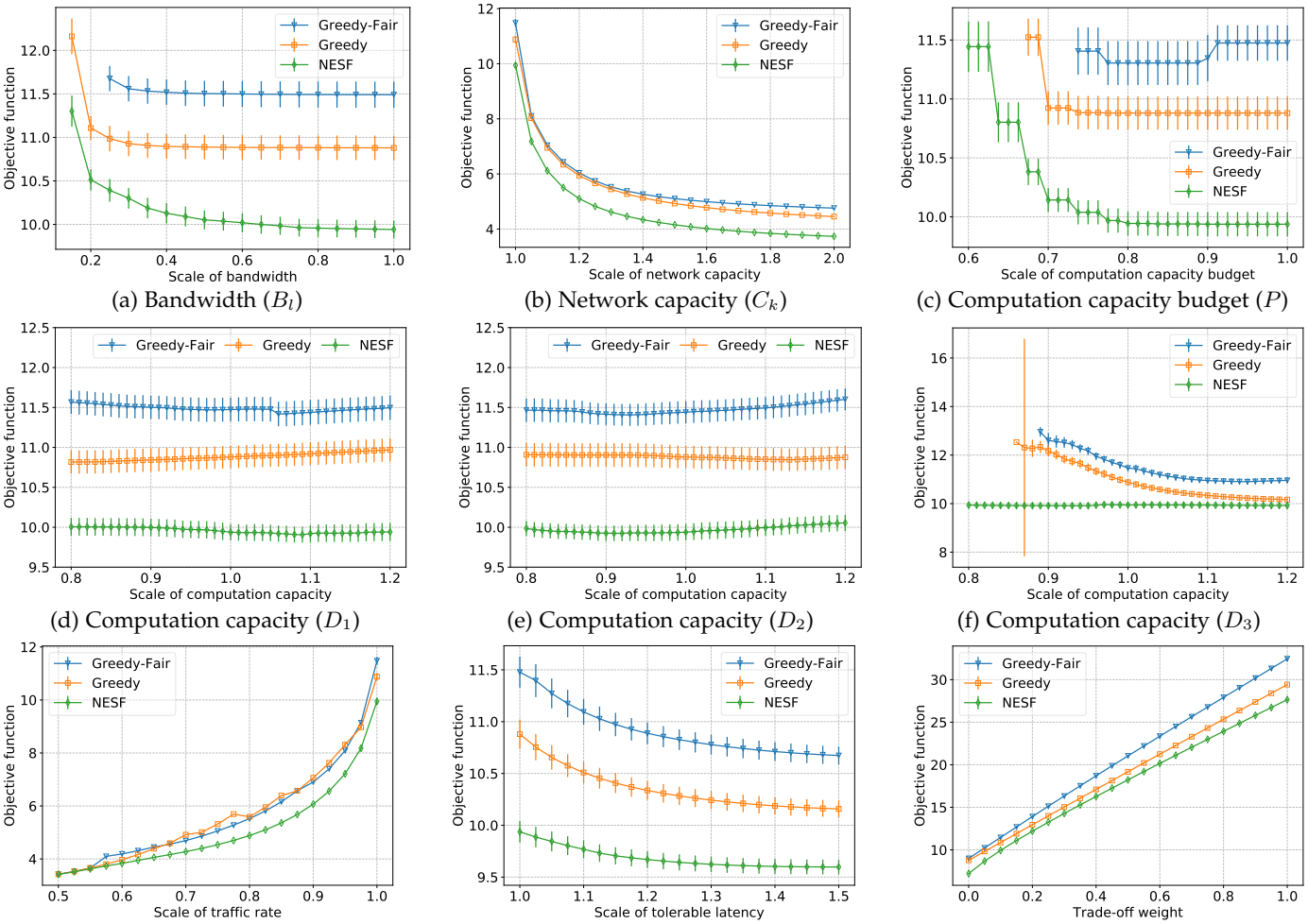
The *Greedy* and *Greedy-Fair* approaches tend to enable computation capacity on more nodes. *Greedy-Fair* also splits each type of traffic following multiple paths. Both aspects result in a higher objective function value.

When increasing the network capacity C_k by the scale factor 1.2, the resulting solutions remain almost the same, except for the allocation of the wireless network capacity and computation capacity.

6.5 Analysis of the heuristic results for larger networks

We investigate the effect of several parameters on the objective function value, with respect to link bandwidth B_l,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3107022, IEEE Transactions on Cloud Computing



(a) Bandwidth (B_l) (b) Network capacity (C_k) (c) Computation capacity budget (P)

(d) Computation capacity (D_1) (e) Computation capacity (D_2) (f) Computation capacity (D_3)

(g) Traffic rate (λ^{kn}) (h) Tolerable latency (τ_n) (i) Trade-off weight (w)

Fig. 10: Numerical results for the large-scale network topology 5(e), 80N120E (averaged over 50 instances).

network capacity C_k, computation capacity D_a and corresponding total budget P, traffic rate λ^{kn}, tolerable latency τ_n and trade-off weight w. We conduct our simulations by scaling one parameter value at a time, starting from the initial values in Table 4. Since the goal is to minimize the weighted sum of total latency and operation cost, lower values for the objective function are preferable.

In Figure 10 we report all results referring to the topology with 80 Nodes and 120 links (Figure 5(e)). All results obtained considering the other topologies in Figure 5 are available here[3] and show similar trends.

6.5.1 Effect of the link bandwidth B_l

Figure 10(a) illustrates the variation of the objective function value (costs w.r.t. latency and computation) versus the link bandwidth $B_l, \forall l \in \mathcal{L}$, the values of which are scaled with respect to its initial ones in Table 4 from 0 to 1.0 with a step of 0.05. In all cases, the problem instance is unfeasible below a certain threshold bandwidth value. As B_l increases above the threshold, the cost value achieved by each approach decreases and converges to a smaller value, i.e., around 9.7 for *NESF* (achieved at 0.9), 10.84 for *Greedy* at 0.3 and 11.48 for *Greedy-Fair* at 0.4. In all cases, *NESF* performs the best among all the approaches, with the following gains: around 11% to *Greedy* and 16% to *Greedy-Fair*. *Greedy* and *Greedy-Fair* show little flexibility to the variation of link bandwidth.

6.5.2 Effect of the wireless network capacity C_k

Figures 10(b) demonstrates the variation of the objective function value with respect to the wireless network capacity $C_k, \forall k \in \mathcal{K}$, scaled with respect to the initial values reported in Table 4 from 1.0 to 2.0, which corresponds to the case in which the wireless network shows a capacity comparable to the one of the internal network links. When C_k increases, the objective function value obtained by each approach decreases quite fast (more than 2 times) and converges to a specific value. For *NESF*, the cost decreases from 9.70 and converges to 3.73; *Greedy* and *Greedy-Fair* exhibit close performance, i.e., *Greedy* from 10.84 to 4.45, *Greedy-Fair* from 11.48 to 4.76. *NESF* still has the best performance among all the approaches, with consistent gaps: around 16% to *Greedy* and up to 22% for *Greedy-Fair*. This trend reflects the strong effect of the wireless network capacity increase on the minimization of the overall system cost and performance.

6.5.3 Effect of the computation capacity budget P

Figures 10(c) shows the trend of the objective function value at the variation of the computation capacity budget P, whose value is scaled with respect to the initial one in Table 4 from 0.5 to 1.0 with a step of 0.0125. Clearly, a low power budget challenges the optimization approach that must ensure the available computation capacity is always

within this budget. The figure shows that each heuristic has a limit budget value below which it is unable to find a feasible solution (0.738 for *Greedy-Fair*, 0.675 for *Greedy* and 0.60 for *NESF*). Thus, *NESF* is the most resilient in this case. As P increases, the cost values obtained by *NESF* and *Greedy* monotonically decrease like staircases, and finally fast converge to specific points, i.e., 9.70 for *NESF* and 10.84 for *Greedy*. The staircase pattern is due to the fact that the optimal solution remains constant when P varies in a small range, and the decreasing trend is also consistent with the real world case. However, the cost value for *Greedy-Fair* exhibits an opposite trend. This is due to its strategy that tries to use the maximum number of nodes that the budget P can cover, and distribute the traffic load on all of them. This scheme, thus, results in a waste of computation capacity and cost increase in some situations. Finally, *NESF* still achieves the best performance, with the following gaps: around 11% to *Greedy* and 16% to *Greedy-Fair*.

6.5.4 Effect of the computation capacity D_a

Figures 10(d), 10(e), and 10(f) illustrate the variations of the objective function value with respect to the three levels of computation capacity D_a, which are scaled from 0.8 to 1.2 w.r.t. the initial values in Table 4 with a step of 0.01, still keeping the relation $D_1 < D_2 < D_3$. In Figures 10(d) and 10(e), the objective function values obtained by the three approaches show very small variation when the computation capacity is scaled. In Figure 10(f), there is a clear decreasing trend for the objective function values achieved by both *Greedy* and *Greedy-Fair*. The reason is that many edge nodes are enabled with the D_3 computation level, and the increased D_3 capacity reduces much of the total latency while not adding much operation cost. The objective function value achieved by *NESF*, on the other hand, almost does not change. To summarize, *NESF* could provide better and more stable solutions, compared with the other approaches.

6.5.5 Effect of the traffic rate λ^{kn}

Figure 10(g) shows the objective function value variation versus the traffic rate. Values $\lambda^{kn}, kn \in \mathcal{K} \times \mathcal{N}$ are scaled from 0.5 to 1.0 with respect to the initial value in Table 4, with a step of 0.025. As traffic λ^{kn} increases, the objective function values for all the approaches increase. We observe that $NESF$ is characterized by a smooth curve, which indicates stability in the solving processing, while both *Greedy* and *Greedy-Fair* exhibit larger fluctuations. When the scale is $\leqslant 0.55$, i.e., the traffic rate is relatively low, the cost values for all the approaches are the same since the best configuration, i.e., locally computing of the traffic, is easily identified by all of them. After that point, *NESF* exhibits a better performance with a clear gap (around 14%) with respect to the other approaches.

6.5.6 Effect of the tolerable latency τ_n

Figure 10(h) illustrates the objective function value with respect to the tolerable latency $\tau_n, n \in \mathcal{N}$ scaled from 1.0 to 1.5 on the initial value in Table 4. When τ_n increases, the objective function values obtained by all the approaches decrease and converge to specific points, i.e., around 9.48 for *NESF*, 10.15 for *Greedy*, and finally 10.64 for *Greedy-Fair*. Parameter τ_n serves in our model as an upper bound (see constraint (18)), and limits the solution space. In fact, with a low τ_n value, the feasible solution set is smaller and the total cost increases, and vice versa. Finally, *NESF* performs the best, with the following gaps: around 7% with respect to *Greedy*, and 11% to *Greedy-Fair*.

We further considered more stringent scenarios where we extended the scaling range of the tolerable latency, τ_n, from 0.75 to 1.5. The results are shown in Figure 11, and are related to the *Città Studi* topology (see Figure 6(c)) and show that, when latency requirements are very stringent (the left part in these figures) the total cost of the network planned to accommodate such stringent requirements sharply increases (see Figure 11(c)). Please also note that, for some of these extreme values of the scaling parameter, the *Greedy* and *Greedy-Fair* benchmark algorithms were unable to find a feasible solution, while our proposed heuristics (*NESF*) is always able to find a solution.

TABLE 5: Impact of the weight w (solution computed by the *NESF* heuristic).

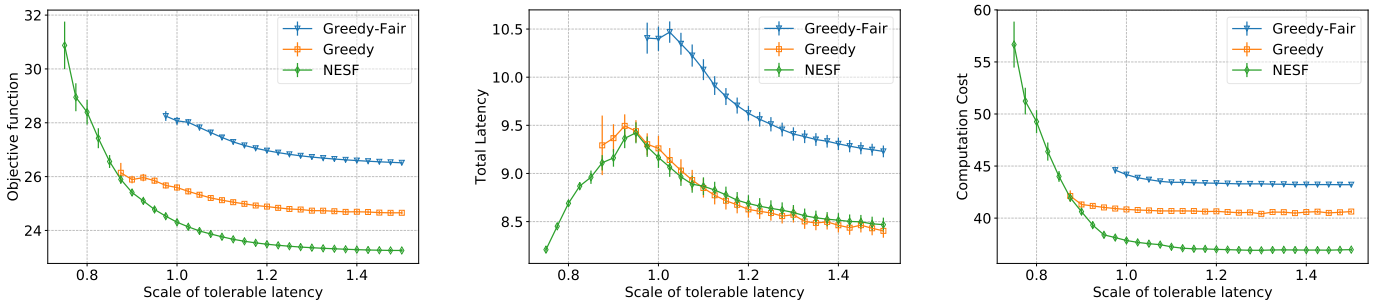
	w	$T + wJ$	T	J
Scaling link bandwidth B_l (factor 0.6)	0.003	8.15	8.01	47.76
	0.1	12.58	8.43	41.57
	0.4	24.35	9.18	37.94
Scaling network capacity C_k (factor 1.5)	0.003	2.21	2.07	47.08
	0.1	6.60	2.86	37.47
	0.4	17.72	2.92	37.00

6.5.7 Effect of the trade-off weight w

This parameter permits to express, in the objective function computation, the relevance of the overall operation cost with respect to the total latency experienced by users. Lower values of w correspond to a lower relevance of the operation cost w.r.t. latency. In Figure 10(i) w is changed from 0 to 1.0 with a step of 0.05. When $w = 0$, the optimization focuses almost exclusively on the total latency. As w increases, the objective function values increase almost linearly for all the approaches. The *NESF* algorithm still achieves the best performance, with gaps around 7% with respect to *Greedy* and 16% w.r.t. *Greedy-Fair*.

Hereafter we present (Table 5) numerical results obtained in the "Città Studi" topology, to illustrate the impact of the trade-off weight w. Following the setting of the weight parameter w discussed in Section 4.3, which permits to privilege the optimization of the network cost J or the delay T, we obtain in this scenario (based on the parameters values), $w_L \approx 0.003$ and $w_U \approx 0.4$. For simplicity, we select three values for w (viz., 0.003, 0.1, 0.4) to give different priorities to the overall latency and planning cost.

Let us analyze the results for scaling network capacity C_k as an example. If we set $w = 0.003$, thus giving priority in the optimization to the minimization of the experienced overall latency T, we see that such value is, in average, 2.07, while the cost of the planned network J is 47.08. In this case, we tend to plan costlier networks but we can satisfy more stringent latency requirements of users. If on the other hand we set $w = 0.4$, thus privileging cost minimization and then reducing latency as second step, we observe that, in average, the latency T is 2.92 while



(a) Objective function (b) Total latency (c) Total network cost

Fig. 11: Scaling tolerable latency τ_n from 0.75 to 1.5, *Città Studi* topology.

the average cost of the planned network J is 37.00. By comparing these two extreme situations we observe that the latency increases of 41%, passing from the first scenario to the second, while in parallel the cost reduces of about 21%. Finally, Figure 12 shows for completeness the whole set of results, that is, the objective function value for the three w settings considered in the previous Table, and for all C_k scaling factors.

6.5.8 Robustness analysis

In the same scenario illustrated in Section 3, we further quantify the robustness of our proposed model and algorithms. To this aim, we increase the traffic from one ingress node ($\lambda^{n5,t2}$) first from 35 to $36Gb/s$ and then from 35 to $40Gb/s$. In both cases, the original scenario, with $\lambda^{n5,t2} = 35Gb/s$, is denoted by the symbol "$\circ$" in Tables 6 and 7, while the changed one is denoted by "$*$".

We compare the solutions computed by three approaches, where *Optimal* solves the problem optimally, *NESF* is the solution provided by our heuristic, $\circ$*NESF* represents the solution computed for the original instance ($\circ$) by directly applying it to the changed one ($*$). The *Margin* row is computed as $\circ$*NESF* - *NESF*. We observe that $\circ$*NESF* can directly provide a feasible solution also for the modified scenario with $\lambda^{n5,t2} = 36Gb/s$, very close to the original one in terms of objective function value.

In the second scenario, since traffic increases more consistently (from 35 to 40 Gb/s), we consider a further approach (named *M*$\circ$*NESF*) to avoid the infeasibility that can be experienced when applying directly, as $\circ$*NESF* does, the solution computed for the original instance ($\circ$) to the changed one ($*$). Indeed, all allocation and routing solutions taken for the original problem are still valid (including decisions c^{kn}, b_i^{kn}, α_i^{kn}, β_i^{kn} and also routing path $\mathcal{R}_i^{kn}$), and we just need to re-optimize planning decisions of computation capacity levels δ_i^a. This permits to avoid infeasibility and to obtain very good solutions: in this scenario the objective function of *Optimal* is 2.415, *NESF* 2.479 and *M*$\circ$*NESF* 2.524, just 1.8% higher than *NESF*.

6.6 Computing Time

Figure 13 compares the average computing time of the proposed approaches under all considered network topologies. The computing time for $\mathcal{P}1$ is shown only for the smallest topology and it is already significantly larger than the others. For the tree-shaped network topology (Figure 5(c)), all approaches are able to obtain the solution very fast, in less than $10s$. This is due to the fact that routing optimization is indeed trivial in such topology. The computing time is ordered as: *Greedy*<*NESF*<*Greedy-Fair*. When considering standard deviation, the order is: *NESF*<*Greedy*<*Greedy-Fair*, and this shows the stability of our proposed approach in the solving process. As for the network topology with 100 nodes and 150 edges (a general large scale network), *NESF* is able to obtain a good solution in around $100s$, and remains below this value in the other considered cases. This gives us an indication that the network management component can periodically run *NESF* as a response to changes in the network or in the incoming traffic, and optimize nodes computation capacities and routing paths accordingly. This is a key feature for providing the necessary QoS levels in next-generation mobile network architectures and for updating it dynamically.

TABLE 6: Robustness analysis for instance *10N20E* ($\circ$: original, $\lambda^{n5,t2} = 35Gb/s$; $*$: changed scenario with $\lambda^{n5,t2} = 36Gb/s$).

	$T + wJ$		T		J		Computing time (s)	
	$\circ$	$*$	$\circ$	$*$	$\circ$	$*$	$\circ$	$*$
Optimal	2.249	2.256	1.049	1.056	12.0	12.0	38463	48521
NESF	2.277	2.281	0.977	0.981	13.0	13.0	1.307	1.169
$\circ$*NESF*	-	2.318	-	1.018	-	13.0	-	0.291
Margin		0.037		0.037		0		

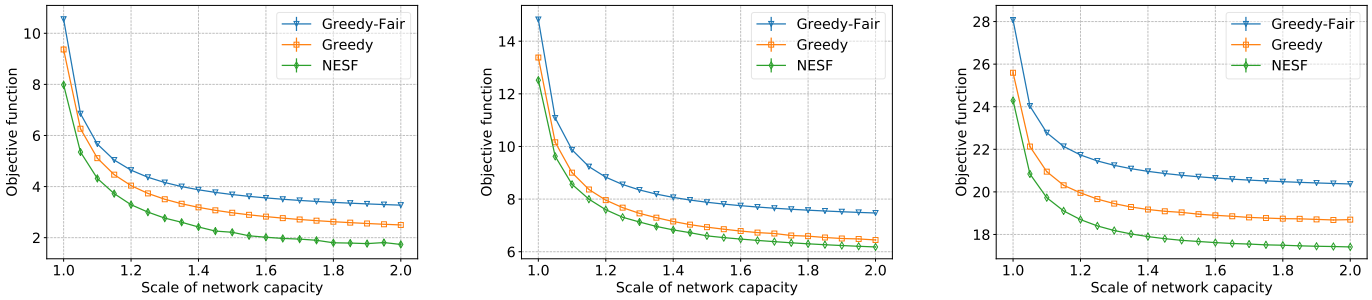
TABLE 7: Robustness analysis for instance *10N20E* ($\circ$: original, $\lambda^{n5,t2} = 35Gb/s$; $*$: changed scenario with $\lambda^{n5,t2} = 40Gb/s$).

	$T + wJ$		T		J		Computing time (s)	
	$\circ$	$*$	$\circ$	$*$	$\circ$	$*$	$\circ$	$*$
Optimal	2.249	2.415	1.049	1.115	12.0	13.0	38463	581077
NESF	2.277	2.479	0.977	0.979	13.0	15.0	1.307	1.273
$\circ$*NESF*	directly applying $\circ$ solution to $*$: *Infeasible*							
M$\circ$*NESF*	-	2.524	-	1.224	-	13.0	-	0.339
Margin		0.045		0.245		-2		

7 Related Work

Several works have been recently published on the resource management problem in a MEC environment; most of them consider a single mobile edge cloud at the ingress node and do not account for its connection to a larger edge cloud network [20–22]. The following of this section provides a short overview on the various areas that are relevant to the problem we consider. As discussed in the **Summary** part, ours is the first approach that considers at the same time multiple aspects related to the configuration of an edge cloud network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3107022, IEEE Transactions on Cloud Computing



(a) $w = 0.003$ (b) $w = 0.1$ (c) $w = 0.4$

Fig. 12: Scaling network capacity C_k under different weight w settings.

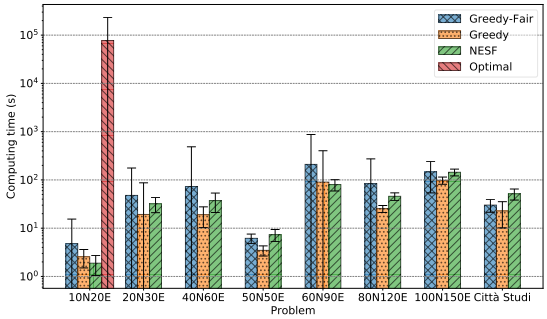


Fig. 13: Computing time.

Network planning: The network planning problem in a MEC/Fog/Cloud context tackles the problems concerning nodes placement, traffic routing and computation capacity configuration. The authors in [23] propose a mixed integer linear programming (MILP) model to study cloudlet placement, assignment of access points (APs) to cloudlets and traffic routing problems, by minimizing installation costs of network facilities. The work in [8] proposes a MILP model for the problem of fog nodes placement under capacity and latency constraints. [11] presents a model to configure the computation capacity of edge hosts and adjust the cloud tenancy strategy for dynamic requests in cloud-assisted MEC to minimize the overall system cost.

Service/content placement: The service and content placement problems are considered in several contexts including, among others, micro-clouds, multi-cell MEC etc. The work in [24] studies the dynamic service placement problem in mobile micro-clouds to minimize the average cost over time. The authors first propose an offline algorithm to place services using predicted costs within a specific look-ahead time-window, and then improve it to an online approximation one with polynomial time-complexity. An integer linear programming (ILP) model is formulated in [25] for serving the maximum number of user requests in edge clouds by jointly considering service placement and request scheduling. The edge clouds are considered as a pool of servers without any topology, which have shareable (storage) and non-shareable (communications, computation) resources. Each user is also limited to use one edge server. In [26], the authors extend the work in [25] by separating the time scales of the two decisions: service placement (per frame) and request scheduling (per slot) to reduce the operation cost and system instability. In [27], the authors study the joint service placement and request routing problem in multi-cell MEC networks to minimize the load of the centralized cloud. No topology is considered for the MEC networks. A randomized rounding (RR) based approach is proposed to solve the problem with a provable approximation guarantee for the solution, i.e., the solution returned by RR is at most a factor (more than 3) times worse than the optimum with high probability. However, although it offers an important theoretical result, the guarantee provided by the RR approach is only specific to the formulated optimization problem. [28] studies the problem of service entities placement for social virtual reality (VR) applications in the edge computing environment. [29] analyzes the mixed-cast packet processing and routing policies for service chains in distributed computing networks to maximize network throughput.

The work in [30] studies the edge caching problem in a Cloud RAN (C-RAN) scenario, by jointly considering the resource allocation, content placement and request routing problems, aiming at minimizing the system costs over time. [31] formulates a joint caching, computing and bandwidth resources allocation model to minimize the energy consumption and network usage cost. The authors consider three different network topologies (ring, grid and a hypothetical US backbone network, US64), and abstract the fixed routing paths from them using the OSPF routing algorithm.

Cloud activation/selection: The cloud activation and selection problems are studied as a way to handle the configuration of computation capacity in a MEC environment. The authors in [32] design an online optimization model for task offloading with a sleep control scheme to minimize the long term energy consumption of mobile edge networks. The authors use a Lyapunov-based approach to convert the long term optimization problem to a per-slot one. No topology is considered for the MEC networks. [33] proposes a model to dynamically switch on/off edge servers and cooperatively cache services and associate users in mobile edge networks to minimize energy consumption. [34] jointly optimizes the active base station set, uplink and downlink beamforming vector selection, and computation capacity allocation to minimize power consumption in mobile edge networks. [35] proposes a model to minimize a weighted sum of energy consumption and average response time in MEC networks, which jointly considers the cloud selection and routing problems. A population game-based approach is designed to solve the optimization problem.

Network slicing: The authors in [36] study the resource allocation problem in network slicing where multiple resources have to be shared and allocated to verticals (5G

end-to-end services). [37] formulates a resource allocation problem for network slicing in a cloud-native network architecture, which is based on a utility function under the constraints of network bandwidth and cloud power capacities. For the slice model, the authors consider a simplified scenario where each slice serves network traffic from a single source to a single destination. For the network topology, they consider a 6x6 square grid and a 39-nodes fat-tree.

Other perspectives: Inter-connected datacenters also share some common research problems with the multi-MEC system. The work in [38] studies the joint resource provisioning for Internet datacenters to minimize the total cost, which includes server provisioning, load dispatching for delay sensitive jobs, load shifting for delay-tolerant jobs, and capacity allocation. [39] presents a bandwidth allocation model for inter-datacenter traffic to enforce bandwidth guarantees, minimize the network cost, and avoid potential traffic overload on low cost links.

The work in [40] studies the problem of task offloading from a single device to multiple edge servers to minimize the total execution latency and energy consumption by jointly optimizing task allocation and computational frequency scaling. In [41], the authors study task offloading and wireless resource allocation in an environment with multiple MEC servers. [42] formulates an optimization model to maximize the profit of a mobile service provider by jointly scheduling network resources in C-RAN and computation resources in MEC.

Summary: To the best of our knowledge, our paper is the first to propose a complete approach that encompasses both the problem of *planning* cost-efficient edge networks and *allocating resources*, performing optimal routing and minimizing the total traffic latency of transmitting, outsourcing and processing user traffic, under a constraint of user tolerable latency for each class of traffic. We model accurately both link and processing latency, using non-linear functions, and propose both exact models and heuristics that are able to obtain near-optimal solutions also in large-scale network scenarios, that include hundreds of nodes and edges, as well as several traffic flows and classes.

8 Conclusion and Future Directions

In this paper, we studied the problem of jointly planning and optimizing the resource management of a mobile edge network infrastructure. We formulated an exact optimization model, which takes into accurate account all the elements that contribute to the overall latency experienced by users, a key performance indicator for these networks, and further provided an effective heuristics that computes near-optimal solutions in a short computing time, as we demonstrated in the detailed numerical evaluation we conducted in a set of representative, large-scale topologies, that include both mesh and tree-like networks, spanning wide and meaningful variations of the parameters' set.

We measured and quantified how each parameter has a distinct impact on the network performance (which we express as a weighted sum of the experienced latency and the total network cost) both in terms of strength and form. Traffic rate and network capacity have the stronger effects, and this is consistent with real network cases. Tolerable latency shows an interesting effect: the lower requirements on latency (or equivalently: the higher value of tolerable latency) the system sets, the lower latency and costs the system will have. This information can be useful for network operators to design the network indicators of services. The computation capacity has relatively smaller effect on the network performance, compared with the other parameters. Another key observation that we draw from our numerical analysis is that as the system capacities (including link bandwidth, network capacity and computation capacity budget) increase, the system performance converges to a plateau, which means that increasing the system capacity over a certain level (which we quantify for each network scenario) will have small effectiveness, and on the contrary, it will increase the total system cost.

Finally, we observe that our models can be extended within the theoretical framework of stochastic optimization, which can be used to guarantee robustness of the solution with respect to the uncertainty in the probabilistic description of traffic demands. Possible extensions of our model could further include explicit modeling of resource scaling across clusters, of VM state and storage synchronization as well as IaaS internal traffic across edge facilities.

Acknowledgment

This research was supported by the H2020-MSCA-ITN-2016 SPOTLIGHT under grant agreement No. 722788 and the H2020-ICT-2020-1 PIACERE under grant agreement No. 101000162.

References

[1] W. Xiang, K. Zheng, and X. S. Shen, *5G mobile communications*. Springer, 2017.
[2] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. 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.
[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing–A key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
[4] R. Kannan and C. L. Monma, "On the computational complexity of integer programming problems," in *Optimization and Operations Research*, Springer, 1978, pp. 161–172.
[5] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "Joint Network Slicing and Mobile Edge Computing in 5G Networks," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.
[6] L. Geng, J. Dong, S. Bryant, K. Makhijani, A. Galis, X. de Foy, and S. Kuklinski, "Network Slicing Architecture," IETF, Internet-Draft, Jul. 2017.
[7] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "How should i slice my network? a multi-service empirical evaluation of resource sharing efficiency," in *Proc. of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 191–206.
[8] A. Santoyo-González and C. Cervelló-Pastor, "Latency-aware cost optimization of the service infrastructure placement in 5G networks," *Journal of Network and Computer Applications*, vol. 114, pp. 29–37, 2018.
[9] R. Rokui, S. Homma, X. de Foy, L. M. Contreras, P. Eardley, K. Makhijani, H. Flinck, R. Schatzmayr, A. Tizghadam, C. Janz, and H. Yu, "IETF Network Slice for 5G and its characteristics," IETF, Internet-Draft, Nov. 2020.
[10] K. Sparks, M. Sirbu, J. Nasielski, L. Merrill, K. Leddy, P. Krishnaswamy, W. Johnston, R. Gyurek, B. Daly, M. Bayliss, J. Barnhill, and K. Balachandran, "5G Network Slicing Whitepaper," *FCC Technological Advisory Council, 5G IoT Working Group*, 2018.
[11] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. S. Shen, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Transactions on Cloud Computing*, 2019.

[12] Y. Niu, B. Luo, F. Liu, J. Liu, and B. Li, "When hybrid cloud meets flash crowd: Towards cost-effective service provisioning," in *IEEE INFOCOM*, 2015, pp. 1044–1052.
[13] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.
[14] P. Luong, F. Gagnon, C. Despins, and L.-N. Tran, "Joint virtual computing and radio resource allocation in limited fronthaul green C-RANs," *IEEE Transactions on Wireless Communications*, vol. 17, no. 4, pp. 2602–2617, 2018.
[15] C.-P. Li, J. Jiang, W. Chen, T. Ji, and J. Smee, "5G ultra-reliable and low-latency systems design," in *European Conference on Networks and Communications (EuCNC)*, IEEE, 2017, pp. 1–5.
[16] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 283–294, 2017.
[17] J. Tang, W. P. Tay, T. Q. Quek, and B. Liang, "System cost minimization in cloud RAN with limited fronthaul capacity," *IEEE Trans. on Wireless Commun.*, vol. 16, no. 5, pp. 3371–3384, 2017.
[18] B. Zhuang, D. Guo, and M. L. Honig, "Energy-efficient cell activation, user association, and spectrum allocation in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 823–831, 2016.
[19] P. Erdős and A. Rényi, "On Random Graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.
[20] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.
[21] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.
[22] X. Ma, S. Zhang, W. Li, P. Zhang, C. Lin, and X. Shen, "Cost-efficient workload scheduling in cloud assisted mobile edge computing," in *Quality of Service (IWQoS), IEEE/ACM 25th International Symposium on*, 2017, pp. 1–10.
[23] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1818–1831, 2017.
[24] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, 2016.
[25] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 365–375.
[26] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019, pp. 1279–1287.
[27] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM*, 2019, pp. 10–18.
[28] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM*, 2018, pp. 468–476.
[29] J. Zhang, A. Sinha, J. Llorca, A. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," in *IEEE INFOCOM*, 2018, pp. 1880–1888.
[30] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1751–1767, 2018.
[31] Q. Chen, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "Joint resource allocation for software-defined networking, caching, and computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 274–287, 2018.
[32] S. Wang, X. Zhang, Z. Yan, and W. Wang, "Cooperative edge computing with sleep control under non-uniform traffic in mobile edge networks," *IEEE Internet of Things Journal*, 2018.
[33] Q. Wang, Q. Xie, N. Yu, H. Huang, and X. Jia, "Dynamic Server Switching for Energy Efficient Mobile Edge Networks," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
[34] J. Opadere, Q. Liu, N. Zhang, and T. Han, "Joint Computation and Communication Resource Allocation for Energy-Efficient Mobile Edge Networks," in *IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
[35] B. Wu, J. Zeng, L. Ge, Y. Tang, and X. Su, "A game-theoretical approach for energy-efficient resource allocation in MEC network," in *IEEE International Conference on Communications (ICC)*, 2019.
[36] F. Fossati, S. Moretti, P. Perny, and S. Secci, "Multi-resource allocation for network slicing," *IEEE/ACM Transactions on Networking*, vol. 28(3), pp. 1311–1324, 2020.
[37] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat, "A resource allocation framework for network slicing," in *IEEE INFOCOM*, IEEE, 2018, pp. 2177–2185.
[38] D. Xu, X. Liu, and Z. Niu, "Joint resource provisioning for internet datacenters with diverse and dynamic traffic," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 71–84, 2017.
[39] W. Li, K. Li, D. Guo, G. Min, H. Qi, and J. Zhang, "Cost-minimizing bandwidth guarantee for inter-datacenter traffic," *IEEE Transactions on Cloud Computing*, 2016.
[40] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.
[41] K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, "Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
[42] X. Wang, K. Wang, S. Wu, S. Di, H. Jin, K. Yang, and S. Ou, "Dynamic resource scheduling in mobile edge cloud with cloud radio access network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2429–2445, 2018.

Bin Xiang received his B.S. degree in electronic engineering from Southwest Jiaotong University, Chengdu, China, in 2013. He is currently pursuing the Ph.D. degree at the Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy. His current research interests include mobile edge computing, network planning, and resource allocation.

Jocelyne Elias is with the Department of Computer Science and Engineering (DISI) at University of Bologna since 2019, and she was Associate Professor at Paris Descartes University since 2010. Her main research interests include network optimization, modeling and performance evaluation of networks (Cognitive Radio, Wireless, Virtual and Wired Networks), and the application of Game Theory to resource allocation, spectrum access, and pricing problems.

Fabio Martignon received the Ph.D. degree in telecommunication engineering from the Politecnico di Milano in 2005. He has been full professor at LRI (Laboratory for Computer Science), Paris Sud University, and he is now full professor at University of Bergamo (Italy) and member of Institut Universitaire de France. His research activities include network planning and game theory applications to mobile networking problems, content distribution and adaptive radio networks.

Elisabetta Di Nitto is Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano, where she also earned her Ph.D. in Computer Science. Her current research interests are mainly on software engineering, and in particular, on process support systems, service-centric applications, dynamic software architectures, self-adaptive systems and cloud-based systems.