Resource Calendaring for Mobile Edge Computing: Centralized and Decentralized Optimization Approaches

Bin Xiang[a], Jocelyne Elias[b], Fabio Martignon[c] and Elisabetta Di Nitto[a]

[a] *Politecnico di Milano, Italy*
[b] *University of Bologna, Italy*
[c] *University of Bergamo, Italy*

ARTICLE INFO

Keywords:
Edge Computing
Network Slicing
Network Design
Calendaring
Joint Optimization
ADMM

ABSTRACT

Mobile Edge Computing (MEC) is a key technology for the deployment of next generation (5G and beyond) mobile networks. The computational power it provides at the edge could allow providers to fulfill the requirements of use cases in need of ultra-low latency, high bandwidth, as well as real-time access to the radio network. However, this potential needs to be carefully administered as the edge is certainly limited in terms of computation capability, as opposed to the cloud which holds the promise of a virtually infinite power. MEC nodes, though, could still try to exploit not only their local capacity, but also the one that the neighbor MEC nodes could offer. Considering that the 5G scenario assumes an ultra-dense distribution of MEC nodes, this possibility could be feasible, provided that we find an effective way to carefully allocate the resources available at each edge node.

In this paper, we provide an optimization framework that considers several key aspects of the resource allocation problem with cooperating MEC nodes. We carefully model and optimize the allocation of resources, including computation and storage capacity available in network nodes as well as link capacity. Specifically, our proposed model jointly optimizes (1) the user requests *admission decision* (2) their *scheduling*, also called *calendaring* (3) and *routing* as well as (4) the decision of which nodes will serve such user requests and (5) the amount of processing and storage capacity reserved on the chosen nodes. Both an exact optimization model and an effective heuristic, based on sequential fixing, are provided.

Furthermore, we propose a distributed approach for our problem, based on the Alternating Direction Method of Multipliers (ADMM), so that resource allocation decisions can be made in a distributed fashion by edge nodes with limited overhead.

We perform an extensive numerical analysis in several real-size network scenarios, using real positions for radio access points of a mobile operator in the Milan area. Results demonstrate that the heuristic performs close to the optimum in all considered network scenarios, while exhibiting a low computing time. This provides an evidence that our proposal is an effective framework for optimizing resource allocation in next-generation mobile networks.

1. Introduction

Next generation (5G and beyond) mobile networks are currently being deployed, and need to provide services characterized by ultra-low latency, high bandwidth, as well as real-time access to the radio network. To achieve these goals, *Mobile Edge Computing (MEC)* is envisaged to provide 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. Through this approach, the latency experienced by mobile users when they use specific services can be considerably reduced. However, the computation power that can be offered by a single edge cloud is limited compared to the one available at a remote cloud. This implies that relying exclusively on a single edge cloud for serving user requests is not actually possible, thus resulting in the need to devise approaches that either offload the exceeding traffic to the central cloud or rely on the availability of other close-by edge clouds that could cooperate to help each other. The cooperation of multiple edge clouds is beneficial to the providers to make full use of the edge computation resources, but, at the same time, it requires a careful allocation of edge resources to each user request. Considering

[⋆] This paper extends the article B. Xiang, J. Elias, F. Martignon E. Di Nitto, Resource Calendaring for Mobile Edge Computing in 5G Networks, accepted at IEEE ICC 2021.

bin.xiang@polimi.it (B. Xiang); jocelyne.elias@unibo.it (J. Elias); fabio.martignon@unibg.it (F. Martignon); elisabetta.dinitto@polimi.it (E.D. Nitto)
ORCID(s):

that 5G networks will be likely built in an ultra-dense manner and the edge clouds attached to 5G base stations will also be massively deployed and connected to each other in a specific mesh topology, this approach appears to be feasible and certainly deserves a specific analysis. It requires the adoption of proper strategies that, on the one side, guarantee the user requirements associated to each request are met and, on the other side, ensure the operator's resources are not depleted but used in an optimized manner. Meeting both the expectations of end users and operators is not straightforward as they are somewhat conflicting with each other. In fact, while from the user viewpoint, requests must be served within the expected time boundaries, from the provider's perspective, serving a request should be sufficiently profitable and not causing the need to exclude other more profitable requests. These requirements result in the need to develop an approach to control multiple aspects, including the decision to admit or not a request, the scheduling of admitted requests in order to fulfill the users' requirements, the assignment of requests to a proper serving node and the subsequent allocation of the required resources, the proper routing of the requests toward their servants.

As discussed in more details in Section 2, the literature presents approaches that handle some of the aspects mentioned above [1–11]. However, to the best of our knowledge, there is no approach tackling all above aspects at the same time.

Our approach is centered on an optimization framework (an exact model as well as an efficient heuristic approach based on sequential fixing) that considers all key aspects of the resource allocation problem in the context of Mobile Edge Computing, by carefully modeling and optimizing the allocation of network resources including computation and storage capacity available in network nodes as well as link capacity. Specifically, our proposed model and heuristics jointly optimize (1) the *admission decision* (which user requests are admitted and served by the network, based on the profit they can potentially generate with respect to the required resources for serving demands), (2) the *scheduling* of admitted user requests, also called *calendaring* (taking into account the flexibility that some users exhibit in terms of starting and ending time tolerated for the required services), (3) the *routing* of these user requests, (4) the decision of which nodes will serve them as well as (5) the amount of processing and storage capacity reserved on the chosen nodes that serve such user requests, with the objective of maximizing the operator's profit.

Besides the classical optimization approach, we propose also a "multi-agent" framework where the resources in the Mobile Edge Network are managed in a coordinated and decentralized way. The goal of each agent is to dynamically allocate network resources, reacting to (possibly local) network changes in a prompt and effective way. This could help reducing the experienced latency, thus permitting to satisfy in a more effective way demanding services. Choosing a distributed approach can also have advantages since edge nodes may be owned and managed by different entities/operators (and hence a centralized approach might be not suitable), and obtaining a centralized solution could be infeasible or require a long computing time in large scale scenarios.

In our approach each agent solves the optimization problem in a distributed fashion, with limited overhead, by relying on an implementation of the Alternating Direction Method of Multipliers [12]. This is a well-established optimization tool used to decompose a problem into multiple small subproblems that can be solved iteratively. We extend it using a *weighting* approach especially tailored to our scheduling problem.

This paper extends the work presented in [13] by strengthening the evaluation of the optimization approach, which now includes realistic network topologies, and by defining and experimenting with the distributed solution approach, based on ADMM.

We provide clear quantitative insights regarding the structure of the underlying computational infrastructure, and we compare our proposed model and heuristic to a greedy approach, which provides a benchmark for our solutions. We perform a thorough performance analysis of the proposed model and heuristics using real-size network scenarios and real radio access points positions of a mobile network operator (Vodafone Italy) in the Milan area. Numerical results demonstrate that our proposed model captures several important aspects of Mobile Edge Computing architectures. Furthermore, the proposed heuristics perform close to the optimum in all considered network scenarios, with a very short computing time, thus representing a very promising solution for the design of efficient and cost-effective mobile networks.

To summarize, our paper makes the following contributions:

- A mathematical model that captures key aspects of Mobile Edge Computing architectures.
- An efficient heuristics, based on a Sequential-Fixing approach [11, 14, 15].
- A fully distributed algorithm based on ADMM, which permits to solve our problem with resource allocation decisions made directly by edge nodes.
- A thorough numerical evaluation performed in several realistic, large scale topologies. We make all topology datasets publicly available in a repository.

Table 1
Comparison of related works in various aspects of resource management.

Reference	Admission	Scheduling	Slicing	Offloading	Routing	Placement	Approach	Scenario
[1]	√		√				Decentralized	Flat MEC
[2]				√			Centralized	Flat MEC
[3]				√		√	Centralized	Fog
[4]				√			Centralized	Multi-layer MEC
[5]				√	√		Centralized	Distributed Computing
[6]	√	√			√		Centralized	SDN
[7]		√			√		Centralized	Data center
[8]		√		√			Decentralized	Flat MEC
[9]		√		√			Decentralized	Flat MEC
[10]		√				√	Centralized	Flat MEC
[16]					√		Decentralized	SDN
[17]		√			√		Centralized	SDN
[18]	√	√					Centralized	Cloud RAN
[19]		√	√				Decentralized	Flat MEC
[20]				√			Decentralized	Flat MEC
Our work	√	√	√	√	√		Both	MEC networks

The paper is organized as follows: Section 2 discusses related work. Section 3 illustrates the problem formulation and the proposed exact optimization model. Section 4 presents the heuristics we devised, based on a sequential-fixing approach. Section 5 illustrates a distributed algorithm we propose, based on ADMM, to solve our problem with resource allocation decisions made directly by edge nodes. The numerical analysis and comparison of the proposed model and heuristics is performed and discussed in Section 6, including real-life network scenarios. Finally, Section 7 concludes the paper.

2. Related Work

In this section we revise works that consider task offloading and calendaring/scheduling issues; we further discuss recent works where the ADMM approach has been applied in mobile networking contexts.

In [2], the authors study a task offloading model considering constraints on task queue lengths to minimize the users' power consumption, while the work in [4] jointly considers task assignment, computing and transmission resources allocation to minimize system latency in a multi-layer MEC context that is, a set of MEC nodes structured in a multi-layer tree-shape network where nodes in a same layer have no interaction. The work in [3] studies task distribution and proactive edge caching in fog networks, that is, networks where edge devices are connected to central nodes, with latency and reliability constraints to minimize the task computing time. The authors in [5] study traffic processing and routing policies for service chains in distributed computing networks to maximize network throughput. These works, however, do not consider the resource scheduling problem. The works in [6, 17, 18] study bandwidth calendaring to allocate network resources and schedule deadline-constrained data transfers, while in [7] the authors study the problem of scheduling and routing deadline-constrained flows in data center networks to minimize the energy consumption. However, the allocation of computing resources is not considered in these works. In [8], the authors study the problem of dispatching and scheduling jobs in edge-cloud system to minimize the job response time; in [9] the authors study online deadline-aware task dispatching and scheduling in edge computing to maximize the number of completed tasks. Finally, the work in [10] proposes a two-time-scale strategy for resource allocation by performing service placement (per frame) and request scheduling (per slot) to reduce the operation cost and system instability. These works, though, do not explicitly consider the routing problem that arises.

ADMM [12] has been recently applied in the mobile networking context. The work in [1] studies a MEC slicing framework named Sl-EDGE which allows network operators to instantiate heterogeneous slice services on edge devices. The authors formulate the edge slicing problem as a mixed-integer linear programming (MILP) model and design a distributed algorithm based on ADMM such that clusters can locally compute slicing strategies. In [19], the authors propose a distributed cross-domain resource orchestration algorithm based on ADMM for dynamic network

Table 2
Summary of used notations.

Parameter	Definition
$\mathcal{K}$	Set of user requests with different demands
$\mathcal{V}$, $\mathcal{E}$	Set of nodes and links in the edge computing network
$\mathcal{T}$	Set of time slots
B_e, ψ_e	Bandwidth of link $e \in \mathcal{E}$ and unit cost
D_v, θ_v	Computation capacity of node $v \in \mathcal{V}$ and unit cost
S_v, ϕ_v	Storage capacity of node v and unit cost
s^k	Source node of request $k \in \mathcal{K}$
α^k, β^k, d^k	Arrival time, deadline and duration of request k
λ^k	Average arrival rate of request k
η^k	Processing density of request k
μ^k	Revenue gained from serving request k
m^k	Amount of storage capacity required to serve request k
Variable	**Definition**
z^{kt}	Whether request k starts at time slot $t \in \mathcal{T}$
q^{kv}	Fraction of request k processed at node v
p_e^{kvt}	Fraction of link e's bandwidth sliced to request q^{kv} at t
r^{kvt}	Fraction of node v's computation capacity sliced to k at t
ρ^{kvt}	Whether node v processes request k at t

slicing in cellular edge computing, while [20] studies the energy-efficient workload offloading problem and propose a low-complexity distributed solution based on consensus ADMM. The work in [16] proposes a distributed algorithm based on ADMM to solve the multi-path fair bandwidth allocation problem in distributed Software Defined Networks (SDN) with the assumption that the paths are pre-computed once and for all and do not change.

To the best of our knowledge, our work is the first one that considers admission decision, scheduling, slicing, offloading and routing all together. The other approaches we presented, instead, focus on specific aspects. To offer an easier comparison with our present work, in Table 1 we provide a summary of the subproblems tackled in each reference (columns from 2 to 7). Moreover, we specify whether the computation of each approach is centralized or decentralized (column 8). Finally, we report the context in which each approach has been developed (column 9). Many of them have been developed in a case that we call *Flat MEC*, in which multiple MEC nodes coexist but are all connected to a core network and are not able to interact with each other. This is different from the "Multi-layer MEC" where, as already mentioned, MEC nodes are structured in a multi-layer tree-shape network and nodes in a same layer have no interaction. Our approach works in a generalization of the Multi-layer MEC one in which all MEC nodes can be connected with each other and no differentiation between layers exist. For this reason, we call it *MEC network*. The approach in [3] has been defined in a fog context, which is essentially equivalent to Flat MEC. The other approaches have been defined in the SDN and distributed/cloud computing contexts.

3. Problem Formulation

In this section we formulate the *resource calendaring* problem, which includes users' requests *admission*, their *scheduling* and *routing*. Our target is to maximize the profit, which is expressed as the difference between the revenue earned by the provider from serving users' requests and the cost incurred from providing computation and storage resources at edge nodes, as well as bandwidth capacity. Table 2 summarizes the notation used throughout this section. For brevity, we simplify expression $\forall k \in \mathcal{K}$ as $\forall k$, and apply the same rule to other set symbols like $\mathcal{V}, \mathcal{E}, \mathcal{T}$, etc. throughout the rest of this paper.

3.1. System Overview

We consider an edge cloud network represented by an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ represents an edge *computing node* having D_v and S_v as *computation* and *storage capacity*, respectively. The two parameters θ_v and ϕ_v denote, respectively, the *cost* of computation and storage capacity of node v. Each *edge* $e \in \mathcal{E}$ corresponds to a network link characterized by its *bandwidth* B_e and its *cost per unit of flow* ψ_e. Let $\mathcal{K}$ denote the set of *requests*, with different types, offered to the network. We regard each type of request as an aggregated communication-computation demand, e.g. web, video, game traffic etc., which has to be accommodated in the network and requires some amount of bandwidth, storage and computation resources. We assume that the calendar (i.e., the arriving time and duration) of the requests for the upcoming period is known. This can be achieved assuming that customers have announced their requirements in advance, or that some history-based prediction tool [21] is used. Since an accurate prediction of network traffic is an important element for network operators, several techniques have been proposed in the literature to enable efficient resource management, traffic engineering and load balancing [22–26]. The aim of these works is to try to predict parameters like the near-term transmission rate of a connection or, in general, network traffic profiles, based on measurements on the past traffic and on service-level agreements established with network users, as a prediction of future traffic distribution; however, the focus of our paper is not to determine the best traffic predictor. The problem of optimal allocation of current and future bandwidth resources is studied in [24] in the context of Traffic Engineering in SDN. Machine Learning approaches have also been proposed to predict the traffic load on the links of a telecom network [22]. Convolutional Neural Networks have been proposed for predicting network traffic in datacenter networks [23]. Finally, when precise prediction is difficult to achieve, online algorithms like the one illustrated in [6] could be considered, to deal with unpredictable incoming demands; these approaches are more fit when an admission decision, scheduling and resource allocation decisions must be taken instantaneously.

We discretize the time horizon into a set $\mathcal{T}$ of equal duration time-slots, where the *slot length* is τ. Each request $k \in \mathcal{K}$ is defined as a tuple $(s^k, \alpha^k, \beta^k, d^k, \lambda^k)$. The parameter s^k is the *source node* of request k; α^k, β^k and d^k define the *arrival time*, the *latest ending time* (deadline) and the *duration* of request k, respectively. Finally, we consider a Poisson process for each request k with an average packet arrival rate λ^k. The arrival and ending times coincide, respectively, with the arrival of the first packet and the departure of the last packet of request k.

A request k could be processed immediately (for delay-sensitive tasks) after its arrival, or scheduled for later (for delay-tolerant tasks). Also, it could be entirely processed on the local edge computing node or split into multiple fractions and processed on other nodes. In any case, it must be completed before the deadline β^k. As an example, Figure 1 shows the arrival time α, deadline β and duration d of requests 1 and 2. Also, it highlights that request 1 is scheduled to be served from time $\xi^1_\star$, delayed (shifted) with respect to α^1 but still compatible with β^1. The ending time for the request will then depend on $\xi^1_\star$, d^1, processing latency, and link latency along the routing path if (some fraction of) the request is offloaded to the neighbor edge computing nodes.

Given a *calendar of requests* $\mathcal{T}$ over a time horizon, the proposed optimization approach must: a) schedule the starting time of each request, b) decide where to compute the requests, and c) route some fractions of the requests when it is necessary to process them on other edge computing nodes, in order to maximize the profit of the provider.

Possible use cases of the above request model can be illustrated by the following examples. For instance, at each edge node we have a stateless servant for a specific request type (it could be, for example, a microservice that is deciding whether some temperature data must trigger an alarm or not) and the network decides where to send a request. When considering streams of data (that is, multiple packets related one with the other), we should have a way to route to the same destination the packets belonging to the same stream. In another case, if we have a computation that requires traffic to be split according to a specific logic (e.g., the typical MapReduce example concerning word counting), the traffic cannot be split by the network in a simple way, but it should go through some application-specific components that decide how to split it.

Note that in above model, we do not explicitly model errors and re-transmissions; a simple way to capture these such features could be to calculate the expectation value of the processing time based on the failure probability, and correspondingly the number of re-transmission attempts. We leave these extensions as futures research issues.

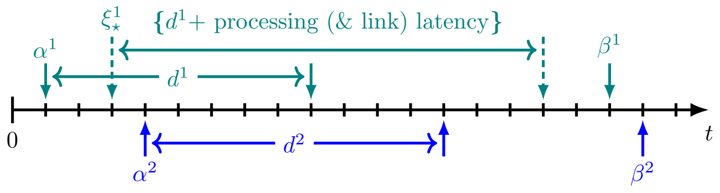


Figure 1: Example of time scheduling of a request.

3.2. Life Cycle of A Request

A given request k arriving at an edge node v at time α^k could be: *i)* rejected, *ii)* processed immediately – this is needed if it is a delay sensitive task – or *iii)* shifted to a future epoch, if it is delay tolerant. To model the fact that the *delayed (shifted)* starting time $\xi_\star^k$ can vary in the time frame $[\alpha^k, \beta^k - d^k]$, we express $\xi_\star^k$ as:

$$\xi_\star^k = \sum\nolimits_{t=\alpha^k}^{\beta^k-d^k} t \cdot z^{kt}, \ \forall k, \tag{1}$$

where z^{kt} is a binary variable that can be 1 at most in one point of time which will correspond to $\xi_\star^k$ for request k. Meanwhile, we have:

$$\sum\nolimits_{t=\alpha^k}^{\beta^k-d^k} z^{kt} \leqslant 1, \ \forall k. \tag{2}$$

When $z^{kt} = 0$ for all possible time slots, this implies that the request is not admitted and, therefore, not scheduled. Note that by changing the inequality constraint (2) to an equality, the edge cloud will be forced to serve all the incoming requests, which may be unfeasible in some cases.

A request can be either processed locally in a computing node or split and offloaded to other edge computing nodes. In the latter case, the *processing latency*, the *storage provisioning* constraints and the *link latency* along a routing path should be taken into account by the calendaring scheme. Considering a node v that is assigned to process a fraction $q^{kv} \in [0,1]$ of request k, the *ending time at* v, denoted by ξ_o^{kv}, can be expressed as:

$$\xi_o^{kv} = \xi_\star^k + d^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil + \left\lceil \frac{T_P^{kv}}{\tau} \right\rceil, \ \forall k, \forall v, \tag{3}$$

where T_L^{kv} and T_P^{kv} are respectively the link latency and processing latency. Note that both $\xi_\star^k$ and ξ_o^{kv} are integer values in the *time slot* set, and τ is the time-slot duration. The ending time of each request depends on the last finished piece, which must be completed before the deadline. Such constraint is expressed as:

$$\max_{v \in \mathcal{V}} \{\xi_o^{kv}\} \leqslant \beta^k, \ \forall k. \tag{4}$$

In the following, we will express the request routing and the two latency components (link and processing latency) in detail.

3.3. Network Routing

We assume that a request can be split into multiple pieces only at its source 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*). Each link $e \in \mathcal{E}$ may carry different request pieces, q^{kv} (remind that q^{kv} is the fraction of request k to be processed at node v). Then, the total flow of request k on link e, f_e^k, can be expressed as the sum of all pieces of k that pass through such link:

$$f_e^k = \sum_{v \in \mathcal{V}:\ e \in \mathcal{R}^{kv}} q^{kv}, \ \forall k, \forall e, \tag{5}$$

where $\mathcal{R}^{kv} \subset \mathcal{E}$ denotes the routing path (set of traversed links) for the partial request q^{kv} from source node s^k to node v. The traffic flow conservation constraint is enforced by:

$$\sum_{e \in \Phi_v^-} f_e^k - \sum_{e \in \Phi_v^+} f_e^k = \begin{cases} q^{kv} - 1, & \text{if } v = s^k, \\ q^{kv}, & \text{otherwise}, \end{cases} \qquad \forall k, \forall v, \tag{6}$$

where Φ_v^- and Φ_v^+ are, respectively, the set of incoming and outgoing links of node v. The fulfillment of this constraint guarantees *continuity* and *acyclicity* for the routing path.

3.4. Link Latency

Let T_L^{kv} denote the *link latency* for routing request k to node v. Each request is routed in a multi-path way, i.e., different pieces of the request may be dispatched to different nodes via different paths. The transmission time of the requests on each link is described by an $M|M|1$ model [27], hence, $\forall k, \forall v$, T_L^{kv} is defined as:

$$T_L^{kv} = \begin{cases} \sum\limits_{e \in \mathcal{R}^{kv}} \frac{1}{p_e^{kv} B_e - q^{kv} \lambda^k}, & \text{if } q^{kv} > 0 \,\&\, v \neq s^k, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

where p_e^{kv} is the fraction of bandwidth capacity sliced for the piece of request (q^{kv}) flowing to node v via link e. The *link latency* is accounted for only if a piece of request k is processed at node v (i.e., $q^{kv} > 0$) and $v \neq s^k$. The following constraint ensures that the flow of request k on each link of the routing path does not exceed the allocated capacity:

$$\begin{cases} q^{kv} \lambda^k < p_e^{kv} B_e, & \text{if } e \in \mathcal{R}^{kv}, \\ p_e^{kv} = 0, & \text{otherwise}, \end{cases} \quad \forall k, \forall v, \forall e. \tag{8}$$

Considering that different requests $k \in \mathcal{K}$ can share the same link at a given time slot, the reservation constraint of a link capacity at any time slot is expressed as:

$$\sum_{k \in \mathcal{K}} \sum_{v \in \mathcal{V}} p_e^{kvt} \leqslant 1, \ \forall e, \forall t, \tag{9}$$

where p_e^{kvt} is the fraction of link e's bandwidth allocated for a piece of request q^{kv} at time slot t. Note that we assume that the reserved bandwidth for each request over its life period does not change in order to provide consistent service guarantees. The superscript t in p_e^{kvt} is used to indicate the life status of the flow. The relation between p_e^{kvt} and p_e^{kv} is given by $p_e^{kvt} = \delta^{kvt} p_e^{kv}$, where δ^{kvt} is a binary variable defined as:

$$\delta^{kvt} = \begin{cases} 1, \ \text{if } \xi_\star^k \leqslant t < \xi_\star^k + d^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil, \\ 0, \ \text{otherwise}, \end{cases} \forall k, \forall v, \forall t. \tag{10}$$

3.5. Processing Latency and Storage Provisioning

When a request cannot be entirely processed locally, we assume that such request can be segmented and processed on different edge computing nodes. Hence, each node can slice its computation capacity to serve several requests coming from different source nodes. Notice that a request k also requires a fixed amount of storage resources m^k on a node v if k is to be processed on that node. Thus, only if both computation and storage resources on a node are sufficient, a request could be processed on that node. Let variable r^{kv} denote the fraction of computation capacity D_v sliced for the piece of request $q^{kv}\lambda^k$. The processing of user requests is also described by an $M|M|1$ model [28, 29]. Let T_P^{kv} denote the *processing latency* of edge computing node v for request k. Then, based on the computational capacity $r^{kv} D_v$ with an amount $q^{kv}\lambda^k$ to be served, $\forall k, \forall v$, T_P^{kv} is expressed as:

$$T_P^{kv} = \begin{cases} \frac{1}{r^{kv} D_v - \eta^k q^{kv} \lambda^k}, & \text{if } q^{kv} > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

where r^{kv} is the fraction of node v's computation capacity sliced to request k, and η^k is the processing density [30] of request k measured in "cycles/bit". In the above equation, when request k is not processed on node v, the latency is set to 0 and, at the same time, no computation resource should be allocated to request k. The corresponding constraint is:

$$\begin{cases} \eta^k q^{kv} \lambda^k < r^{kv} D_v, & \text{if } q^{kv} > 0, \\ r^{kv} = q^{kv} = 0, & \text{otherwise}. \end{cases} \tag{12}$$

q^{kv} and r^{kv} also have to fulfill the consistency constraints:

$$\sum\nolimits_{v \in \mathcal{V}} q^{kv} = \sum\nolimits_{t=\alpha^k}^{\beta^k - d^k} z^{kt}, \ \forall k. \tag{13}$$

Remind that the right hand of equation (13) represents whether a request k is admitted in the system or not. If a request k is rejected by the admission controller, the right hand expression is equal to 0 and $q^{kv} = 0$ is enforced.

Different requests $k \in \mathcal{K}$ may share an edge computing node at a time slot. Thus, the reservation constraint of a node computation capacity at any time slot is modeled as follows:

$$\sum\nolimits_{k\in\mathcal{K}} r^{kvt} \leqslant 1,\ \forall v, \forall t, \tag{14}$$

where r^{kvt} is the fraction of node v's computation capacity allocated for request k at time slot t. We assume that the reserved computation power for each request over its life period will not change due to both the computation scaling overhead and task reconfiguration overhead. The superscript t in r^{kvt} allows us to keep track of the life status of the request. The relation between r^{kvt} and r^{kv} is given by $r^{kvt} = \rho^{kvt} r^{kv}$, where ρ^{kvt} is a binary variable defined as:

$$\rho^{kvt} = \begin{cases} 1, \ \text{if } \xi_\star^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil \leqslant t < \xi_o^{kv}, \\ 0, \ \text{otherwise}, \end{cases} \quad \forall k, \forall t, \forall v, \tag{15}$$

which indicates whether node v processes request k at t. Finally, based on the definition (15), the storage constraint can be expressed as follows, considering that the storage allocated for an admitted request could be released after the end of its serving process:

$$\sum\nolimits_{k\in\mathcal{K}} m^k \rho^{kvt} \leqslant S_v,\ \forall v, \forall t. \tag{16}$$

In the above formulation, the way we model latency and delay is aligned with other approaches in the literature. The work of Ma et al. [27] 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, authors also assume 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 [31]. In [28], 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 [29] assumes that the baseband processing of each Virtual Machine (VM) on each User Equipment packet 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 [32–35] also adopt similar choices concerning the delay modeling.

3.6. Optimization Problem

Our goal in the *resource calendaring* problem is to maximize the profit computed as the total revenue obtained from serving the users' requests minus the network operation costs in terms of computation, storage and bandwidth resources, under the constraints (starting and ending times) of requests coming from different nodes:

$$\begin{aligned} \max_{z^{kt}, q^{kv}, p_e^{kvt}, r^{kvt}, \rho^{kvt}} \quad & \sum_{t\in\mathcal{T}} \sum_{k\in\mathcal{K}} \left\{ \mu^k z^{kt} - \sum_{v\in\mathcal{V}} \left\{ r^{kvt} D_v \theta_v + \rho^{kvt} m^k \phi_v + \sum_{e\in\mathcal{E}} p_e^{kvt} B_e \psi_e \right\} \right\}, \\ \text{s.t.} \quad & (1) \sim (16), \end{aligned} \tag{$\mathcal{P}0$}$$

where μ^k is the revenue gained from serving request k. The variables being optimized, reported under the max operator, are $z^{kt}, q^{kv}, p_e^{kvt}, r^{kvt}, and \rho^{kvt}$. 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 [36]. Since this problem contains the *multi-commodity integer flow problem* as a special case (in fact, in our work flows can be split only at the edge nodes and once admitted they cannot be further split and are therefore routed as integer flows), which is known to be NP-complete [37], it turns out to be NP-hard.

The objective function we have defined in this work is composed of two terms; the first one is the revenue and is calculated as the product of a real coefficient, μ^k, and the admission variable z^{kt}. The second term is the cost incurred by the operator for serving the users' requests; it is calculated as a function of the computation/storage and bandwidth resources, and it directly depends on the total amount of traffic of the request to be served. Our choice of a fixed

Algorithm 1 *Sequential Fixing and Scheduling*

```
1:  Initialize $z^{kt} = 0, \forall k, t$ and profit $\mathcal{O} = 0$ for $(\mathcal{P}.\mathcal{R}1)$;
2:  Set $\hat{b}^{kv} = 0, \hat{r}^{kv} = 0, \hat{p}_e^{kv} = 0$ (in $\mathcal{S}^\star$), $\forall k, v, e$;
3:  Sort $\mathcal{K}$ in descending order as $\mathcal{K}_r$ by $\frac{\mu^k}{d^k \lambda^k m^k \eta^k}, k \in \mathcal{K}$;
4:  for $k \in \mathcal{K}_r$ do
5:      Reset admission $z^{kt} \leqslant 1,\ t \in [\alpha^k, \beta^k - d^k]$;
6:      $F_{k'} \leftarrow check\_overlap(k, k', \mathcal{S}^\star),\ \forall k' \in \mathcal{K} \backslash \{k\}$;
7:      $C_v \leftarrow \max_{k' \in \mathcal{K} \backslash \{k\}} \{\hat{b}^{k'v} F_{k'}\},\ \forall v \in \mathcal{V}$;
8:      $\mathcal{Q}^k, L_v \leftarrow find\_candidates(k, \mathcal{G}, \mathcal{S}^\star, C_v)$;
9:      if $\mathcal{Q}^k \neq \varnothing$ then
10:         $B'_e \leftarrow B_e(1 - \sum_{(k',v) \in \mathcal{K} \times \mathcal{V} | C_v > 0} \hat{p}_e^{k'v}),\ \forall e \in \mathcal{E}$;
11:         Create graph $\mathcal{G}'$ weighted by $B_e'^{-1}$;
12:         for $\mathcal{V}_i \in \mathcal{Q}^k$ do
13:             Set $b^{kv} = 1, r^{kv} \leqslant L_v,\ \forall v \in \mathcal{V}_i$;
14:             Fix route $(\gamma_e^{kv})$ using Dijkstra;
15:             Optimize $(\mathcal{P}.\mathcal{R}1)$ to get profit $\mathcal{O}$ and solution $\mathcal{S}$;
16:             if $\mathcal{O} \geqslant 0$ then break; ▷ $(\mathcal{P}.\mathcal{R}1)$ is feasible
17:     if $\mathcal{O} \geqslant \mathcal{O}^\star$ & $\mathcal{Q}^k \neq \varnothing$ then
18:         Update $\mathcal{O}^\star \leftarrow \mathcal{O},\ \mathcal{S}^\star \leftarrow \mathcal{S}$;
19:         Admit $k$ and allocate resources based on $\mathcal{S}^\star$;
20:         Set $(\mathcal{P}.\mathcal{R}1)$'s lower bound $LB = \mathcal{O}^\star$;
21:     else Reject $k$ (set $z^{kt} = 0, \forall k, t$);
```

revenue per type of request in this paper is three-fold: i) a pricing model (the revenue) which is proportional to the amount of traffic of the user's request and that ignores the type of request may result inappropriate in some scenarios. In fact, each type of request may require different amount of computation/storage/bandwidth resources. ii) The cost defined in the second term of the objective function (which could be perceived as the lower bound of the price to be declared to the user) takes already into account all the aspects mentioned before (in point (i)), including the amount of traffic of the users' requests. iii) Finally, a fixed revenue has been introduced to make the model more flexible and able to admit in the system the requests that can be satisfied with the available resources.

Moreover, we also face the following difficulties: a) routing variables $\mathcal{R}^{kv}$ and request fraction variables q^{kv} are "intertwined": to find the optimal routing, the fraction of each request processed at each node v should be known, and at the same time, to solve the optimal resource allocation for a request, the routing path should be known; b) $(\mathcal{P}0)$ contains indicator functions and constraints, e.g., (7), (10), (15), etc., which cannot be directly and easily processed by most solvers. To deal with the above critical issues, we propose an equivalent reformulation of $(\mathcal{P}0)$, which we call $(\mathcal{P}.\mathcal{R}1)$, that we can efficiently solve with the Branch and Bound method. Intuitively, the reformulation in $(\mathcal{P}.\mathcal{R}1)$ proceeds as follows: (a) we first handle the difficulties related to variables $\mathcal{R}^{kv}$ and the corresponding routing constraints, then (b) we reformulate the link and processing latency constraints (viz., constraints (7) and (11)). Note that, the objective function of $(\mathcal{P}.\mathcal{R}1)$ is the same as that of $(\mathcal{P}0)$, while constraints (4)~(8), (10)~(12) and (15) in $(\mathcal{P}0)$ are reformulated to the constraints (57), (60)~(62) and (67)~(79) in $(\mathcal{P}.\mathcal{R}1)$. For the sake of conciseness, we do not include the reformulation here. The interested readers can refer to Appendix A.

4. Heuristics

To solve our problem in a reasonable computational time, we propose a heuristics named *Sequential Fixing and Scheduling (SFS)* which realizes a good trade-off between admitting "valuable" user requests (i.e., the ones that provide high return to the service provider) and the resources they request in terms of transmission rate, storage and computation. A *greedy approach* is then illustrated as benchmark heuristic.

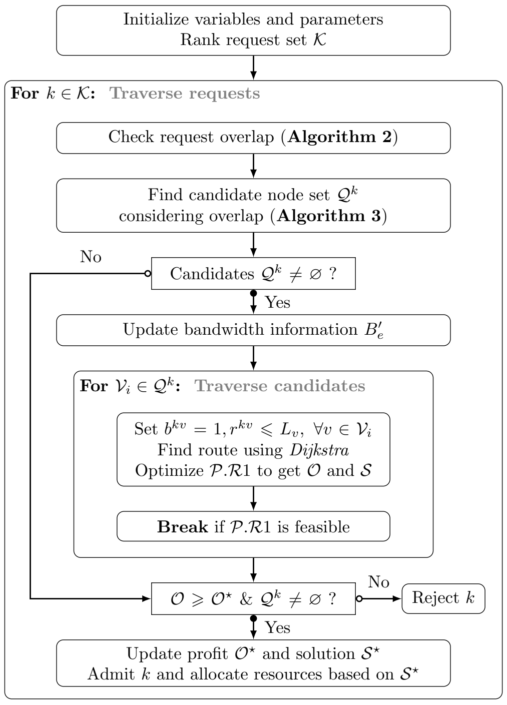

Initialize variables and parameters
Rank request set $\mathcal{K}$
For $k \in \mathcal{K}$: Traverse requests
Check request overlap (Algorithm 2)
Find candidate node set $\mathcal{Q}^k$ considering overlap (Algorithm 3)
No
Candidates $\mathcal{Q}^k \neq \varnothing$?
Yes
Update bandwidth information B'_e
For $\mathcal{V}_i \in \mathcal{Q}^k$: Traverse candidates
Set $b^{kv} = 1, r^{kv} \leqslant L_v, \forall v \in \mathcal{V}_i$
Find route using Dijkstra
Optimize $\mathcal{P}.\mathcal{R}1$ to get $\mathcal{O}$ and $\mathcal{S}$
Break if $\mathcal{P}.\mathcal{R}1$ is feasible
$\mathcal{O} \geqslant \mathcal{O}^\star$ & $\mathcal{Q}^k \neq \varnothing$?
No
Reject k
Yes
Update profit $\mathcal{O}^\star$ and solution $\mathcal{S}^\star$
Admit k and allocate resources based on $\mathcal{S}^\star$

Figure 2: Flowchart of the SFS heuristic.

4.1. Sequential Fixing and Scheduling

SFS is detailed in **Algorithm 1**, and summarized in the flowchart of Figure 2. We first introduce the following auxiliary variables b^{kv} that indicates whether request k is processed on node v and γ_e^{kv} that indicates whether request piece q^{kv} is routed via link e. The hat notation (like $\hat{b}^{kv}$) represents the values of the corresponding variables in the solution set $\mathcal{S}^\star$. We start by sorting all requests in descending order according to the ratio $\frac{\mu^k}{d^k \lambda^k m^k \eta^k}$; this ranking is designed to give a higher weight to requests that generate more revenue and less cost to the operator. Then, we try to define a schedule where we admit as many requests as possible. For each request k, we check whether its activation period overlaps with the one of other requests k' that are already admitted, and in such case we say there is a *conflict*. The overlap value $F_{k'}$ is determined by the function $check\_overlap(\cdot)$ (line 6 of Algorithm 1, details are provided in Section 4.1.1).

Based on $\{F_{k'} | k' \in \mathcal{K} \backslash \{k\}\}$, for each edge node $v \in \mathcal{V}$, we select the maximum $F_{k'}$ for all k' being processed at v, and we identify this overlap value with C_v (line 7). Next, we compute the ordered set $\mathcal{Q}^k$, which contains sets $\mathcal{V}_i$ of best candidate edge nodes to process request k. In doing so, we consider C_v and limit to L_v the computation resource of each node in $\mathcal{V}_i, \forall \mathcal{V}_i \in \mathcal{Q}^k$ (line 8 of Algorithm 1; details in Section 4.1.2). If we successfully find some candidates ($\mathcal{Q}^k \neq \varnothing$), we further update the residual bandwidth B'_e for all links e and create a weighted graph $\mathcal{G}'$ with the reciprocal bandwidth B'^{-1}_e. Then, steps in lines (12-16) are the solution exploration phase based on the set of candidate node groups, which can handle infeasibility situations in the optimization process. More specifically, we select the first $\mathcal{V}_i$ in $\mathcal{Q}^k$ that permits to find a profitable solution ($\mathcal{O} \geqslant 0$) according to the following criteria: we outsource k to the nodes in $\mathcal{V}_i$ and bound the computation resource by setting b^{kv} and r^{kv}. Based on $\mathcal{G}'$, we route each piece of request q^{kv} using the *Dijkstra* algorithm (lines 10-14). After fixing variables b^{kv}, γ_e^{kv} and the constraints related to z^{kt}, r^{kv} in ($\mathcal{P}.\mathcal{R}1$), we start to optimize ($\mathcal{P}.\mathcal{R}1$) to get the profit and the solution denoted, respectively, by $\mathcal{O}$ and $\mathcal{S}$. If ($\mathcal{P}.\mathcal{R}1$) results infeasible in the current setting ($\mathcal{O} < 0$), we reiterate on the other elements of $\mathcal{Q}^k$. If the result of the new optimization improves, we update the current best profit $\mathcal{O}^\star$ and solution $\mathcal{S}^\star$, we hence admit request k and allocate resources to it (including time slots, computation, bandwidth, and storage). We also update the lower bound of ($\mathcal{P}.\mathcal{R}1$) to $LB = \mathcal{O}^\star$ to accelerate the optimization (line 20). Finally, if the result does not improve or no candidate could be found, we reject k and clear its corresponding settings of variables.

Algorithm 2 *Check Overlap*

```
    Input: $k,\ k',\ \mathcal{S}^\star$(solution); Output: $conflict$;
1:  $\alpha = \alpha^{k'},\ \beta = \beta^{k'}$;
2:  if $\mathcal{S}^\star \neq \emptyset$ then $\alpha = \hat{\xi}_\star^{k'},\ \beta = \max_{v\in\mathcal{V}} \hat{\xi}_o^{k'v}$;
3:  $overlap \leftarrow \begin{cases} \beta^k - \alpha, \text{if } \alpha > \alpha^k, \\ \beta - \alpha^k, \text{otherwise;} \end{cases}$
4:  $conflict \leftarrow \min(\frac{\max(overlap,\ 0)}{\min(\beta^k - \alpha^k,\ \beta - \alpha)},\ 1)$;
```

4.1.1. *Check overlap*

The $check\_overlap$ function takes as input k, k' and the partial solution $\mathcal{S}^\star$ computed up to the current point, returns $F_{k'}$ and proceeds as detailed in **Algorithm 2**: i) it initializes two local variables α and β with the starting time $\alpha^{k'}$ and deadline $\beta^{k'}$ of request k', respectively; ii) it verifies if k' is admitted; if yes, it updates, respectively, α and β with the exact starting time $\hat{\xi}_\star^{k'}$ and ending time $\max_{v\in\mathcal{V}} \hat{\xi}_o^{k'v}$ of k' according to the solution $\mathcal{S}^\star$; iii) it computes the (partial) overlapping between k and k' as: $overlap = \beta^k - \alpha$, if $\alpha > \alpha^k$; $overlap = \beta - \alpha^k$, otherwise (a negative value of $overlap$ means no overlapping); iv) Finally, it calculates and returns the maximum relative overlap value $F_{k'}$ between k and k', which is expressed as $\min(\frac{\max(overlap,\ 0)}{\min(\beta^k-\alpha^k,\ \beta-\alpha)},\ 1)$.

Algorithm 3 *Find Candidates*

```
    Input: $k,\ \mathcal{G},\ \mathcal{S}^\star$(solution), $C_v$(conflict);
    Output: $\mathcal{Q}^k$(candidates), $L_v$(limit);
1:  $L_v \leftarrow 1 - \sum_{k'\in\mathcal{K}|C_v > w_c} \hat{r}^{k'v},\ \forall v \in \mathcal{V}$; ▷ $w_c = 0.6$
2:  $\mathcal{V}^s = \{v \in \mathcal{V} \mid C_v \leqslant w_c \mid\mid L_v \geqslant w_l\}$; ▷ $w_l = 0.25$
3:  $\mathcal{Y}_v \leftarrow (-hop(\mathcal{G}, s^k, v), v \notin \{s^{k'} | k' \in \mathcal{K}\}, L_v),\ \forall v \in \mathcal{V}^s$;
4:  Sort $\mathcal{V}^s$ in descending order by $\mathcal{Y}_v$;
5:  $\mathcal{V}_1 = \emptyset,\ D_\Sigma = 0$;
6:  for $v \in \mathcal{V}^s$ do
7:      if $\lambda^k \geqslant w_d D_\Sigma$ then $\mathcal{V}_1 \leftarrow \mathcal{V}_1 \cup \{v\}$; ▷ $w_d = 0.9$
8:      $D_\Sigma \leftarrow D_\Sigma + D_v L_v$;
9:  $\mathcal{Q}^k = (\mathcal{V}_1) \cup (\{v\}, v \in \mathcal{V}^s - \mathcal{V}_1)$;
```

4.1.2. *Find candidates*

Algorithm 3 determines the appropriate subset of edge nodes that can process traffic requests. We first estimate L_v, the remaining computation capacity of each node v, based on $\hat{r}^{k'v}$ in the solution $\mathcal{S}^\star$, verifying that the conflicts are higher than a given threshold w_c ($C_v > w_c$). The threshold w_c reflects the availability of the computation resource for an in-using node in time conflict. Lower w_c values will lead to a less efficient utilization of the computation resources, thus producing a less optimal solution, while higher values could lead to a higher bias in the estimation, therefore leading to an infeasible solution. In the experiments, we choose a proper value $w_c = 0.6$. Then, we define $\mathcal{V}^s$ as the set of nodes v satisfying $C_v \leqslant w_c \mid\mid L_v \geqslant w_l$, where w_l is a threshold on the remaining computation; w_l controls the minimum remaining computation capacity that a candidate should have. Higher w_l values will cause a lower resource utilization. We choose a small value around $w_l = 0.25$ in the experiments. Note that very low values would cause overloading of the computing nodes and lead to an infeasible solution. Hence, $\mathcal{V}^s$ represents the set of nodes that are either in less conflict (for request k) or have enough remaining computation power. For each $v \in \mathcal{V}^s$, we compute three features (denoted by $\mathcal{Y}_v$), i.e., the negative hop distance between s^k and v ($-hop(\mathcal{G}, s^k, v)$), the indicator of whether v is a source node or not ($v \notin \{s^{k'} | k' \in \mathcal{K}\}$) and the estimated left computation capacity L_v (lines 1-3). Based on $\mathcal{Y}_v$, we sort $\mathcal{V}^s$ in *descending* order to give more priority to a node that 1) is closer to the ingress node for k, 2) better not to be a source, and 3) has more remaining computation capacity with respect to other nodes. Then, we try to add nodes to $\mathcal{V}_1$, until $\lambda^k < w_d D_\Sigma$, where D_Σ denotes the estimated total computation capacity that can be used and w_d is a threshold controlling the total required computation capacity. Higher w_d values will make the computation resource utilization more efficient, in cooperation with the solution exploration phase (lines 12-16), while lower values will result in less efficient utilization of computation resources. In the experiments, we set $w_d = 0.9$. Finally, we return the ordered set $\mathcal{Q}^k$ with $\mathcal{V}_1$ at the first place and the left nodes $\mathcal{V}^s - \mathcal{V}_1$ being separately stored as unit sets of backup candidates (lines 5-9). Notice that the values of the thresholds $w_{c/l/d}$ are appropriately chosen based on our experiments.

4.2. Greedy approach

Greedy is a heuristics alternative to SFS that we propose as a benchmark in Section 6. The detailed procedures are listed in Algorithm 4. *Greedy* shares some common steps with *SFS*, but it also exhibits several differences in that it applies different strategies to prioritize requests (line 3) and to search candidate nodes for processing requests (line 6, Algorithm 5); furthermore, it does not consider requests' overlap and the exploration of solutions in case of infeasibility. Note that *Greedy* could solve problems in a short computing time while still obtaining good solutions, as we discuss in Section 6.3. More specifically, it first sorts the requests in ascending order by the priority key $(-\mu^k,\ \alpha^k,\ \beta^k)$ and then tries to schedule them one by one. The sorting considers the revenue μ^k of a request in the first place, the starting time α^k in the second place and finally the deadline β^k in the third (last) place. Then, for each request k, we try to guarantee sufficient computation power by using its closest neighbor nodes. The steps for searching candidate nodes are detailed in Algorithm 5. Compared with the strategy of *SFS* (Algorithm 3), *Greedy* estimates the left computation L_v and the potential set of nodes $\mathcal{V}^s$ (lines 1-2) without considering the request overlap information C_v, and collects candidate nodes until $\lambda^k < w_{greedy} D_\Sigma$, where w_{greedy} is a threshold similar to w_d in *SFS* controlling the required number of computing nodes. The lower w_{greedy} is, the more computation nodes are required. Unlike *SFS*, the remaining neighbor nodes are not used for further solution exploration in case of infeasibility; as a result, higher w_{greedy} values will make the algorithm individuate fewer computing nodes, which may be not sufficient in some cases for processing the requests, leading to less profitable or infeasible solutions. We set $w_{greedy} = 0.6$ in our experiments and evaluate the effect of w_{greedy} on the performance in Section 6.3. Finally, this greedy searching strategy brings a faster computation time, but in general a less optimal solution.

Algorithm 4 *Greedy Algorithm*

```
 1: Initialize $z^{kt} = 0, \forall k, t$ and profit $\mathcal{O} = 0$ for $(\mathcal{P}.\mathcal{R}1)$;
 2: Set $\hat{b}^{kv} = 0, \hat{r}^{kv} = 0, \hat{p}_e^{kv} = 0$ (in $\mathcal{S}^\star$), $\forall k, v, e$;
 3: Sort $\mathcal{K}$ in descending order as $\mathcal{K}_r$ by priority key $(-\mu^k,\ \alpha^k,\ \beta^k)$;
 4: for $k \in \mathcal{K}_r$ do
 5:     Reset admission $z^{kt} \leqslant 1,\ t \in [\alpha^k, \beta^k - d^k]$;
 6:     $\mathcal{V}^k, L_v \leftarrow find\_greedy\_candidates(k, \mathcal{G}, \mathcal{S}^\star)$;
 7:     if $\mathcal{V}^k \neq \emptyset$ then
 8:         $B'_e \leftarrow B_e(1 - \sum_{(k',v)\in\mathcal{K}\times\mathcal{V}} \hat{p}_e^{k'v}),\ \forall e \in \mathcal{E}$;
 9:         Create graph $\mathcal{G}'$ weighted by $B_e'^{-1}$;
10:         Set $b^{kv} = 1, r^{kv} \leqslant L_v,\ \forall v \in \mathcal{V}^k$;
11:         Fix route $(\gamma_e^{kv})$ using Dijkstra;
12:         Optimize $(\mathcal{P}.\mathcal{R}1)$ to get profit $\mathcal{O}$ and solution $\mathcal{S}$;
13:     if $\mathcal{O} \geqslant \mathcal{O}^\star$ & $\mathcal{Q}^k \neq \emptyset$ then
14:         Update $\mathcal{O}^\star \leftarrow \mathcal{O},\ \mathcal{S}^\star \leftarrow \mathcal{S}$;
15:         Admit $k$ and allocate resources based on $\mathcal{S}^\star$;
16:         Set $(\mathcal{P}.\mathcal{R}1)$'s lower bound $LB = \mathcal{O}^\star$;
17:     else Reject $k$ (set $z^{kt} = 0, \forall k, t$);
```

4.3. Complexity Analysis

In this Section we provide a complexity analysis for both the SFS and Greedy approaches.

SFS: Referring to Algorithm 1, the computation complexity comes mostly from the following two aspects: i) the *for* loop with $|\mathcal{K}_r|$ iterations (line 4), together with the nested *for* loop with $|\mathcal{Q}^k|$ iterations (line 12), and ii) the optimization of the reduced problem $(\mathcal{P}.\mathcal{R}1)$ in the nested loop (line 15). Regarding the nested *for* loop, $\mathcal{Q}^k$ represents the set of candidate nodes groups picked from the neighbor nodes. In the worst case, it is $\max\{|\mathcal{Q}^k|\} = |\mathcal{V}|$. However, the actual value of $|\mathcal{Q}^k|$ is, in practice, much lower than $|\mathcal{V}|$ due to the few neighbor nodes that are typically needed for request

Algorithm 5 *Find Greedy Candidates*

```
    Input: $k$, $\mathcal{G}$, $\mathcal{S}^\star$(solution);
    Output: $\mathcal{V}^k$(candidates), $L_v$(limit);
1:  $L_v \leftarrow 1 - \sum_{k' \in \mathcal{K}} \hat{r}^{k'v}$, $\forall v \in \mathcal{V}$;
2:  $\mathcal{V}^s = \{v \in \mathcal{V} \mid L_v > 0\}$;
3:  $\mathcal{Y}_v \leftarrow (-hop(\mathcal{G}, s^k, v), L_v)$, $\forall v \in \mathcal{V}^s$;
4:  Sort $\mathcal{V}^s$ in descending order by $\mathcal{Y}_v$;
5:  $\mathcal{V}^k = \emptyset$, $D_\Sigma = 0$;
6:  for $v \in \mathcal{V}^s$ do
7:      if $\lambda^k \geqslant w_{greedy} D_\Sigma$ then $\mathcal{V}^k \leftarrow \mathcal{V}^k \cup \{v\}$;
8:      $D_\Sigma \leftarrow D_\Sigma + D_v L_v$;
```

offloading and the *break* step (line 16) in the nested loop. Regarding the optimization step, the reduced problem ($\mathcal{P}.\mathcal{R}1$) is still an MINLP problem, hence NP-hard, since it still contains both integer and continuous variables (e.g., scheduling and offloading variables) as well as nonlinear constraints (e.g., latency components). Therefore, for simplicity, we denote the complexity of optimizing ($\mathcal{P}.\mathcal{R}1$) as $O(f(|\mathcal{K}|, \mathcal{G}, |\mathcal{T}|))$, where $f(\cdot)$ is a complexity function related to the number of requests, network topology and the time horizon in the problem. Then, the complexity of optimizing ($\mathcal{P}.\mathcal{R}1$) is $O(f(1, \mathcal{G}, |\mathcal{T}|))$ since, in each iteration, only one request is involved. Therefore, the final complexity of *SFS* can be expressed as $O(|\mathcal{K}||\mathcal{V}|f(1, \mathcal{G}, |\mathcal{T}|))$. Note that the other steps in Algorithm 1 are basic computations with negligible complexity compared to the $O(f(1, \mathcal{G}, |\mathcal{T}|))$.

Greedy approach: Compared with Algorithm 1 (*SFS*), Algorithm 4 has different request prioritizing and candidates seeking strategies, but no request overlap checking and exploration of solutions for infeasible situations. Therefore, regarding its computation complexity, the main difference is that *Greedy* does not have the solution exploration phase. Based on a similar complexity analysis as illustrated above for *SFS*, the complexity of *Greedy* can be expressed as $O(|\mathcal{K}|f(1, \mathcal{G}, |\mathcal{T}|))$.

5. Distributed Resource Allocation

In this Section we illustrate a distributed algorithm for our problem such that resource allocation decisions can be made directly by edge nodes with limited overhead. To this aim we adopt the Alternating Direction Method of Multipliers [12], which is a well-established optimization tool used to decompose a problem into multiple small subproblems that can be solved iteratively in a distributed fashion.

ADMM was originally introduced for solving convex problems with fast convergence properties. Recently, it has been used as an heuristic to approximately solve some nonconvex problems as well, but in this case convergence may not be guaranteed [38, 39]. In the following, we propose a *weighted* ADMM algorithm to decompose and solve our scheduling problem, where a weighting strategy is exploited to balance the priorities of various decision variables in the optimization model. In practice, we observe that the weighting strategy is important for the fast convergence of ADMM in all the network scenarios we considered. For the sake of simplicity and clarity, we first present the ADMM formulation without our weighting scheme, then we present the weighted ADMM algorithm, built with few reformulations.

Table 3 summarizes the main notations used in this section. In the following, notations like $\boldsymbol{x}_{i+1}^{kv}$, $\boldsymbol{y}_{i+1}^{kv}$ and $\boldsymbol{u}_{i+1}^{kv}$, represent the value of the corresponding variables $\boldsymbol{x}^{kv}$, $\boldsymbol{y}^{kv}$ and $\boldsymbol{u}^{kv}$ at iteration $i+1$ of the ADMM algorithm. Notations ${}_z\boldsymbol{y}^{kv}$, ${}_q\boldsymbol{y}^{kv}$, ${}_r\boldsymbol{y}^{kv}$, ${}_\rho\boldsymbol{y}^{kv}$ and ${}_p\boldsymbol{y}^{kv}$ represent the sub-blocks of vector $\boldsymbol{y}^{kv}$, which correspond to the five main decision variables denoted by symbols z, q, r, ρ and p, respectively. The same rule is applied to vector $\boldsymbol{u}^{kv}$.

5.1. ADMM Formulation

Let us recall that our problem aims at scheduling and computing multiple requests from different ingress nodes in a network with a specific (and generic) topology, where each request can be split and offloaded to the nearby edge computing nodes. Hence, we can split the problem across both (i) the request set $\mathcal{K}$ and (ii) the node set $\mathcal{V}$.

To derive the ADMM formulation of problem ($\mathcal{P}.\mathcal{R}1$), we proceed to reformulate it in 3 main steps, illustrated in Figure 3. Specifically, we first select the main decision variables and the corresponding constraints that *bind* variables

Table 3
Summary of main notations for the ADMM Formulation.

Symbol	Definition
$\mathbf{A}$	Diagonal matrix for weight balance in ADMM
$\boldsymbol{x}^{kv}$	Vector of the main decision variables $(\mathring{\boldsymbol{z}}^{kv}, \mathring{q}^{kv}, \boldsymbol{r}^{kv}, \mathring{\boldsymbol{\rho}}^{kv}, \boldsymbol{p}^{kv})^{\mathsf{T}}$ w.r.t. request k on node v
$\boldsymbol{y}^{kv}$	Variable *copy* of $\boldsymbol{x}^{kv}$ for splitting the *sharing* constraints
$\boldsymbol{u}^{kv}$	Vector of the scaled Lagrange multipliers $({}_{z}\boldsymbol{u}^{kv}, {}_{q}\boldsymbol{u}^{kv}, {}_{r}\boldsymbol{u}^{kv}, {}_{\rho}\boldsymbol{u}^{kv}, {}_{p}\boldsymbol{u}^{kv})^{\mathsf{T}}$
$\mathring{f}_{kv}(\boldsymbol{x}^{kv})$	Negative latent profit (cost - revenue) function w.r.t. request k on node v
σ	Penalty parameter in the augmented Lagrangian function
ζ^{primal}, ζ^{dual}	Primal and dual residuals of ADMM
ϵ^{primal}, ϵ^{dual}	Primal and dual feasibility tolerances of ADMM

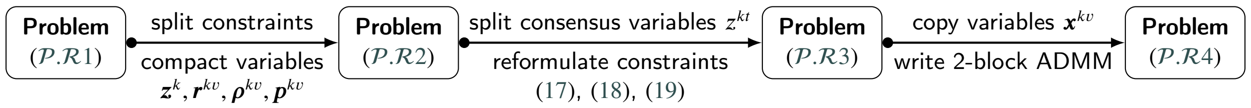

Problem $(\mathcal{P}.\mathcal{R}1)$
split constraints
compact variables $\boldsymbol{z}^k, \boldsymbol{r}^{kv}, \boldsymbol{\rho}^{kv}, \boldsymbol{p}^{kv}$
Problem $(\mathcal{P}.\mathcal{R}2)$
split consensus variables z^{kt}
reformulate constraints (17), (18), (19)
Problem $(\mathcal{P}.\mathcal{R}3)$
copy variables $\boldsymbol{x}^{kv}$
write 2-block ADMM
Problem $(\mathcal{P}.\mathcal{R}4)$

Figure 3: Flowchart of ADMM formulation with main procedures.

with respect to k and v, which hinder the distributed optimization of the problem. The main variables are written in compact form for the sake of clarity and the convenience of derivation. This step transforms the problem into $(\mathcal{P}.\mathcal{R}2)$. However, $(\mathcal{P}.\mathcal{R}2)$ contains *consensus* variable z^{kt}, which means that, given a request k, this information is shared by the edge nodes where the request k might be offloaded to. This also prevents the distributed optimization. Then, to handle this issue, we split the consensus variable z^{kt} by introducing a local *copy* of z^{kt} on each possible edge node and reformulating the corresponding constraints (17), (18) and (19), and transform the problem into $(\mathcal{P}.\mathcal{R}3)$. In $(\mathcal{P}.\mathcal{R}3)$, both the objective function and variables are *splittable* across the index k and v, while the main constraints for the requests and capacities still bind the variables together. Finally, to solve this, we *copy* the main decision variables (denoted by $\boldsymbol{x}^{kv}$), introduce a penalty function based on the constraints, and then formulate the problem into $(\mathcal{P}.\mathcal{R}4)$ based on 2-block ADMM. $(\mathcal{P}.\mathcal{R}4)$ is a standard ADMM formulation which can be applied for distributed optimization.

In the following, we present in detail the 3 steps of reformulations. Firstly, we write the main decision variables in the following compact form:

$$\boldsymbol{z}^k = (z^{kt})^{\mathsf{T}}_{t\in\mathcal{T}},\ \boldsymbol{r}^{kv} = (r^{kvt})^{\mathsf{T}}_{t\in\mathcal{T}},\ \boldsymbol{\rho}^{kv} = (\rho^{kvt})^{\mathsf{T}}_{t\in\mathcal{T}},\ \boldsymbol{p}^{kv} = (p_e^{kvt})^{\mathsf{T}}_{t\in\mathcal{T},e\in\mathcal{E}},$$

where $(\cdot)^{\mathsf{T}}$ is the transpose of a given matrix. Let us define:

$$f_{kv}(\boldsymbol{r}^{kv}, \boldsymbol{\rho}^{kv}, \boldsymbol{p}^{kv}) = \sum_{t\in\mathcal{T}} \left\{ r^{kvt} D_v \theta_v + \rho^{kvt} m^k \phi_v + \sum_{e\in\mathcal{E}} p_e^{kvt} B_e \psi_e \right\}.$$

Then, the resource calendaring optimization problem (auxiliary variables, like b^{kv}, γ_e^{kv}, etc., are ignored here), can be reformulated as follows:

$$\min \sum_{k\in\mathcal{K}} \left\{ \sum_{v\in\mathcal{V}} f_{kv}(\boldsymbol{r}^{kv}, \boldsymbol{\rho}^{kv}, \boldsymbol{p}^{kv}) - \sum_{t\in\mathcal{T}} \mu^k z^{kt} \right\}, \tag{$\mathcal{P}.\mathcal{R}2$}$$

$$\text{s.t. } \xi_\star^{kv} = \sum\nolimits_{t=\alpha^k}^{\beta^k-d^k} t \cdot z^{kt}, \qquad \forall k, \forall v, \tag{17}$$

$$\sum\nolimits_{t=\alpha^k}^{\beta^k-d^k} z^{kt} \leqslant 1, \qquad \forall k, \tag{18}$$

$$\sum\nolimits_{v\in\mathcal{V}} q^{kv} = \sum\nolimits_{t=\alpha^k}^{\beta^k-d^k} z^{kt}, \qquad \forall k, \tag{19}$$

$$\sum\nolimits_{k\in\mathcal{K}} r^{kvt} \leqslant 1, \qquad \forall v, \forall t, \tag{20}$$

$$\sum\nolimits_{k\in\mathcal{K}} m^k \rho^{kvt} \leqslant S_v, \qquad \forall v, \forall t, \tag{21}$$

$$\textstyle\sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}} p_e^{kvt} \leqslant 1, \qquad \forall e, \forall t, \tag{22}$$

$$(\boldsymbol{r}^{kv}, \boldsymbol{\rho}^{kv}, \boldsymbol{p}^{kv})^{\mathsf{T}} \in \mathcal{X}^{kv}, \qquad \forall k, \forall v, \tag{23}$$

$$\mathcal{X}^{kv} : \text{rest constraints of } (\mathcal{P}.\mathcal{R}1), \tag{24}$$

where $\mathcal{X}^{kv}$ is the feasible set defined by the rest constraints of ($\mathcal{P}.\mathcal{R}1$) presented in Appendix A. Note that these constraints are splittable across the index k and v. For the sake of clarity, we change the previous auxiliary variable $\xi_\star^k$ (recall that $\xi_\star^k$ is the start time of request k) to $\xi_\star^{kv}$ in our model, where $\xi_\star^{kv}$ can be regarded as the duplicated information of starting time for request k on node v. Since $\xi_\star^{kv}, \forall v$, refers to the same starting time for request k expressed in constraint (17), problem $\mathcal{P}.\mathcal{R}2$ is equivalent to ($\mathcal{P}.\mathcal{R}1$). Constraints (17), (18) and (19) perform the *admission* of a request, formulated by the *consensus* variable z^{kt}, while (20), (21) and (22) represent the *capacity* constraints for processing, storage and bandwidth, which respectively *couple* the variables $\{\boldsymbol{r}^{kv}\}, \{\boldsymbol{\rho}^{kv}\}$ and $\{\boldsymbol{p}^{kv}\}$.

To enable the distributed optimization, we need to split the *consensus* variable z^{kt} and reformulate the corresponding constraints (17), (18) and (19). To do this, we first introduce $\mathring{z}^{kvt}$ as a local copy of z^{kt} for each request on each possible edge node, and add the consensus constraint $\mathring{z}^{kvt} = z^{kt}, \forall k, \forall v, \forall t$. Constraints (17) and (18) can thus be rewritten as:

$$\xi_\star^{kv} = \textstyle\sum_{t=\alpha^k}^{\beta^k-d^k} t \cdot \mathring{z}^{kvt}, \; \forall k, \forall v, \tag{25}$$

$$\textstyle\sum_{t=\alpha^k}^{\beta^k-d^k} \mathring{z}^{kvt} \leqslant 1, \qquad \forall k, \forall v. \tag{26}$$

Then, substituting z^{kt} for $\frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}} \mathring{z}^{kvt}$ (since $\mathring{z}^{kvt} = z^{kt}, \forall k, v, t$) into constraint (19) and introducing an auxiliary variable $\mathring{q}^{kv}$ defined as:

$$\mathring{q}^{kv} = q^{kv} - \frac{1}{|\mathcal{V}|}\sum_{t=\alpha^k}^{\beta^k-d^k} \mathring{z}^{kvt}, \; \forall k, \forall v, \tag{27}$$

we can reformulate constraint (19) to $\sum_{v\in\mathcal{V}} \mathring{q}^{kv} = 0, \forall k$.

Let $\mathring{\boldsymbol{z}}^{kv} = (\mathring{z}^{kvt})_{t\in\mathcal{T}}^{\mathsf{T}}$, $\mathring{\boldsymbol{\rho}}^{kv} = m^k \boldsymbol{\rho}^{kv}$, and $\boldsymbol{x}^{kv} := (\mathring{\boldsymbol{z}}^{kv}, \mathring{q}^{kv}, \boldsymbol{r}^{kv}, \mathring{\boldsymbol{\rho}}^{kv}, \boldsymbol{p}^{kv})^{\mathsf{T}}$. We further reformulate the objective function with respect to $\boldsymbol{x}^{kv}$. To do this, we define a function $\mathring{f}_{kv}(\boldsymbol{x}^{kv})$ as:

$$\mathring{f}_{kv}(\boldsymbol{x}^{kv}) = f_{kv}(\boldsymbol{r}^{kv}, \mathring{\boldsymbol{\rho}}^{kv}/m^k, \boldsymbol{p}^{kv}) - \frac{\mu^k}{|\mathcal{V}|}\sum_{t\in\mathcal{T}} \mathring{z}^{kvt} \tag{28}$$

Based on above, the problem $\mathcal{P}.\mathcal{R}2$ can be rewritten as

$$\min \sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}} \mathring{f}_{kv}(\boldsymbol{x}^{kv}), \tag{$\mathcal{P}.\mathcal{R}3$}$$

$$\text{s.t.} \quad \mathring{\boldsymbol{z}}^{kv} = \boldsymbol{z}^k, \qquad \forall k, \forall v, \tag{29}$$

$$\textstyle\sum_{v\in\mathcal{V}} \mathring{q}^{kv} = 0, \qquad \forall k, \tag{30}$$

$$\textstyle\sum_{k\in\mathcal{K}} \boldsymbol{r}^{kv} \leqslant \mathbf{1}, \qquad \forall v, \tag{31}$$

$$\textstyle\sum_{k\in\mathcal{K}} \mathring{\boldsymbol{\rho}}^{kv} \leqslant \boldsymbol{S}_v, \qquad \forall v, \tag{32}$$

$$\textstyle\sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}} \boldsymbol{p}^{kv} \leqslant \mathbf{1}, \tag{33}$$

$$\boldsymbol{x}^{kv} \in \mathring{\mathcal{X}}^{kv}, \qquad \forall k, \forall v, \tag{34}$$

where $\mathring{\mathcal{X}}^{kv}$ is the new feasible set defined by the constraints in (24) together with (25), (26) and (27). ($\mathcal{P}.\mathcal{R}3$) is more compact and clearer than ($\mathcal{P}.\mathcal{R}2$) in terms of the decision variables and constraints, and the objective function is now expressed as the sum of independent functions $\mathring{f}_{kv}(\boldsymbol{x}^{kv})$. However, we still have several constraints including the request scheduling constraints (29) and (30) as well as the capacities provisioning (31), (32) and (33), which bind the variables together with respect to the index k for the request and v for the edge node. This makes it difficult to compute the optimal solution in a distributed way. For this reason, in the following, we further reformulate ($\mathcal{P}.\mathcal{R}3$) into a 2-block ADMM form, as defined in [12], which can instead be optimized in a distributed manner.

To this aim, let us now define an indicator function:

$$\mathcal{I}_{\mathcal{A}}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{A}, \\ +\infty & \text{otherwise,} \end{cases} \tag{35}$$

where $\mathcal{A}$ is a feasible set defined by constraints (29)~(33).

By introducing an auxiliary variable $\boldsymbol{y}^{kv} := ({}_z\boldsymbol{y}^{kv}, {}_q\boldsymbol{y}^{kv}, {}_r\boldsymbol{y}^{kv}, {}_\rho\boldsymbol{y}^{kv}, {}_p\boldsymbol{y}^{kv})^\top$ as a copy of $\boldsymbol{x}^{kv}$ and setting $\boldsymbol{y} = \{\boldsymbol{y}^{kv}\}_{(k,v)\in\mathcal{K}\times\mathcal{V}}$, we have the following reformulation:

$$\min_{\{\boldsymbol{x}^{kv}\},\boldsymbol{y}} \quad \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}} \mathring{f}_{kv}(\boldsymbol{x}^{kv}) + \mathcal{I}_{\mathcal{A}}(\boldsymbol{y}), \tag{$\mathcal{P}.\mathcal{R}4$}$$

$$\begin{aligned} \text{s.t.} \quad & \boldsymbol{x}^{kv} = \boldsymbol{y}^{kv}, && \forall k, \forall v, \\ & \boldsymbol{x}^{kv} \in \mathring{\mathcal{X}}^{kv}, && \forall k, \forall v. \end{aligned} \tag{36}$$

5.2. ADMM Solution

To derive the solution for the optimization ($\mathcal{P}.\mathcal{R}4$), we first write the **augmented Lagrangian** function as follows:

$$\mathcal{L}_\sigma(\{\boldsymbol{x}^{kv}\}, \boldsymbol{y}; \{\mathring{\boldsymbol{u}}^{kv}\}) = \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}} \left\{ \mathring{f}^{kv}(\boldsymbol{x}^{kv}) + \langle \mathring{\boldsymbol{u}}^{kv}, \boldsymbol{x}^{kv} - \boldsymbol{y}^{kv} \rangle + \frac{\sigma}{2} ||\boldsymbol{x}^{kv} - \boldsymbol{y}^{kv}||^2 \right\} + \mathcal{I}_{\mathcal{A}}(\boldsymbol{y}), \tag{37}$$

where $\mathring{\boldsymbol{u}}^{kv}$ is the vector of Lagrange multipliers (or dual variables) and the component $\langle \mathring{\boldsymbol{u}}^{kv}, \boldsymbol{x}^{kv} - \boldsymbol{y}^{kv} \rangle$ is the inner product. The last term ($\frac{\sigma}{2}||\boldsymbol{x}^{kv} - \boldsymbol{y}^{kv}||^2$) is a *penalty* term and σ is a positive coefficient. The rationale behind introducing such penalty term is that the augmented dual function can be shown to be differentiable under rather mild conditions on the original problem [12].

Then, based on the 2-block **ADMM**, we can write the solution (in a scaled form) as:

$$\begin{cases} \boldsymbol{x}^{kv}_{i+1} := \underset{\boldsymbol{x}^{kv}}{\operatorname{argmin}} \left\{ \mathring{f}^{kv}(\boldsymbol{x}^{kv}) + \frac{\sigma}{2} ||\boldsymbol{x}^{kv} - \boldsymbol{y}^{kv}_i + \boldsymbol{u}^{kv}_i||^2 \right\}, \text{ s.t. } \boldsymbol{x}^{kv} \in \mathring{\mathcal{X}}^{kv}, & (38) \\ \boldsymbol{y}_{i+1} := \underset{\boldsymbol{y}}{\operatorname{argmin}} \left\{ \mathcal{I}_{\mathcal{A}}(\boldsymbol{y}) + \frac{\sigma}{2} \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}} ||\boldsymbol{y}^{kv} - \boldsymbol{x}^{kv}_{i+1} - \boldsymbol{u}^{kv}_i||^2 \right\}, & (39) \\ \boldsymbol{u}^{kv}_{i+1} := \boldsymbol{u}^{kv}_i + \boldsymbol{x}^{kv}_{i+1} - \boldsymbol{y}^{kv}_{i+1}, & (40) \end{cases}$$

where i is the iteration index and $\boldsymbol{u}^{kv} := ({}_z\boldsymbol{u}^{kv}, {}_q\boldsymbol{u}^{kv}, {}_r\boldsymbol{u}^{kv}, {}_\rho\boldsymbol{u}^{kv}, {}_p\boldsymbol{u}^{kv})^\top = \mathring{\boldsymbol{u}}^{kv}/\sigma$ is the scaled dual variable. The steps for updating $\boldsymbol{x}^{kv}_{i+1}$ and $\boldsymbol{u}^{kv}_{i+1}$ can be carried out independently in parallel for each $(k, v) \in \mathcal{K}\times\mathcal{V}$. Notice that the ***y**-update* requires solving a problem having $|\mathcal{K}||\mathcal{V}|$ blocks of variables.

In ***y**-update*, since the l_2 norm $||\boldsymbol{y}^{kv} - \boldsymbol{x}^{kv}_{i+1} - \boldsymbol{u}^{kv}_i||^2$ can be written in a separate form, and constraints (29)~(33) are independent of each other, we could split the indicator function $\mathcal{I}_{\mathcal{A}}(\boldsymbol{y})$ and separately update the copying variables ${}_z\boldsymbol{y}^{kv}, {}_q\boldsymbol{y}^{kv}, {}_r\boldsymbol{y}^{kv}, {}_\rho\boldsymbol{y}^{kv}$ and ${}_p\boldsymbol{y}^{kv}$. The update solution for each sub-item of $\boldsymbol{y}^{kv}$ is written as follows (for the detailed derivation, we refer the interested readers to Appendix B):

$$\boldsymbol{y}^{kv}_{i+1} = \begin{bmatrix} {}_z\boldsymbol{y}^{kv}_{i+1} \\ {}_q\boldsymbol{y}^{kv}_{i+1} \\ {}_r\boldsymbol{y}^{kv}_{i+1} \\ {}_\rho\boldsymbol{y}^{kv}_{i+1} \\ {}_p\boldsymbol{y}^{kv}_{i+1} \end{bmatrix} := \begin{bmatrix} \overline{\boldsymbol{z}}^k_{i+1} + {}_z\overline{\boldsymbol{u}}^k_i \\ \mathring{q}^{kv}_{i+1} - \overline{q}^k_{i+1} \\ \boldsymbol{r}^{kv}_{i+1} - \overline{\boldsymbol{r}}^v_{i+1} + \mathbf{P}_{[\mathbf{0},\mathbf{1}/|\mathcal{K}|]}(\overline{\boldsymbol{r}}^v_{i+1} + {}_r\overline{\boldsymbol{u}}^v_i) \\ \mathring{\boldsymbol{\rho}}^{kv}_{i+1} - \overline{\boldsymbol{\rho}}^v_{i+1} + \mathbf{P}_{[\mathbf{0},\boldsymbol{S}_v/|\mathcal{K}|]}(\overline{\boldsymbol{\rho}}^v_{i+1} + {}_\rho\overline{\boldsymbol{u}}^v_i) \\ \boldsymbol{p}^{kv}_{i+1} - \overline{\boldsymbol{p}}_{i+1} + \mathbf{P}_{[\mathbf{0},\mathbf{1}/(|\mathcal{K}||\mathcal{V}|)]}(\overline{\boldsymbol{p}}_{i+1} + {}_p\overline{\boldsymbol{u}}_i) \end{bmatrix}, \tag{41}$$

where $\mathbf{P}_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{e}) = (\min\{\max\{\boldsymbol{a}_j, \boldsymbol{e}_j\}, \boldsymbol{b}_j\})_{j=1}^{\dim(\boldsymbol{e})}$ denotes the projection of $\boldsymbol{e}$ on box $[\boldsymbol{a}, \boldsymbol{b}]$, and $\overline{\boldsymbol{z}}^k_{i+1}, \overline{q}^k_{i+1}, \overline{\boldsymbol{r}}^v_{i+1}, \overline{\boldsymbol{\rho}}^v_{i+1}, \overline{\boldsymbol{p}}_{i+1}$ are the average value for each item of $\boldsymbol{x}^{kv}_{i+1}$. A similar representation is done for $\boldsymbol{u}^{kv}_{i+1}$.

Note that the step for updating $\boldsymbol{x}_{i+1}^{kv}$ can be carried out in parallel for each $(k, v) \in \mathcal{K}\times\mathcal{V}$. After gathering the average information of $\boldsymbol{x}_{i+1}^{kv}$ and $\boldsymbol{u}_{i}^{kv}$, the updates for $\boldsymbol{y}_{i+1}^{kv}$ can be independently carried out in parallel for each $(k, v) \in \mathcal{K}\times\mathcal{V}$. Finally, the updates are applied in parallel to $\boldsymbol{u}_{i+1}^{kv}$.

5.3. ADMM Convergence

To check the convergence of ADMM, we first compute the primal and dual residuals (denoted by ζ_{i+1}^{primal} and ζ_{i+1}^{dual} respectively):

$$\begin{cases} \zeta_{i+1}^{primal} &= ||\boldsymbol{x}_{i+1}^{kv} - \boldsymbol{y}_{i+1}^{kv}||, & (42)\\ \zeta_{i+1}^{dual} &= \sigma||\boldsymbol{y}_{i+1}^{kv} - \boldsymbol{y}_{i}^{kv}||. & (43)\end{cases}$$

These values converge to zero as the ADMM algorithm progresses [12]. A recommended stopping criterion of ADMM is defined in [12] as follows:

$$\zeta_{i+1}^{primal} \leqslant \epsilon_{i+1}^{primal} \text{ and } \zeta_{i+1}^{dual} \leqslant \epsilon_{i+1}^{dual}, \tag{44}$$

where ϵ_{i+1}^{primal} and ϵ_{i+1}^{dual} are the feasibility tolerances whose expressions are:

$$\begin{cases} \epsilon_{i+1}^{primal} &= \epsilon^{abs}\sqrt{n} + \epsilon^{rel}\max\{||\boldsymbol{x}_{i+1}^{kv}||, ||\boldsymbol{y}_{i+1}^{kv}||\}, & (45)\\ \epsilon_{i+1}^{dual} &= \epsilon^{abs}\sqrt{n} + \epsilon^{rel}\sigma||\boldsymbol{u}_{i+1}^{kv}||, & (46)\end{cases}$$

where $n = \mathsf{dim}(\boldsymbol{x}_{i+1}^{kv})$ is the dimensionality of $\boldsymbol{x}_{i+1}^{kv}$, and ϵ^{abs} and ϵ^{rel} are the absolute and relative tolerances respectively. In practice, we can set $\epsilon^{abs} = \epsilon^{rel} = 10^{-5}$. The choice of these values depends on the application scenario and the scale of variable values.

5.4. Weighted ADMM

In the augmented Lagrangian function (37), the variables in $\boldsymbol{x}^{kv} := (\mathring{\boldsymbol{z}}^{kv}, \mathring{q}^{kv}, \boldsymbol{r}^{kv}, \mathring{\boldsymbol{\rho}}^{kv}, \boldsymbol{p}^{kv})^{\mathsf{T}}$ have different dimensions, i.e., $\mathring{q}^{kv}$ has a dimension of 1, $\mathring{\boldsymbol{z}}^{kv}, \boldsymbol{r}^{kv}$ and $\mathring{\boldsymbol{\rho}}^{kv}$ have the same dimension of $|\mathcal{T}|$, while the dimension of $\boldsymbol{p}^{kv}$ is $|\mathcal{E}||\mathcal{T}|$. Therefore, the penalty term in Lagrangian function (37) equivalently assigns different weights for the variables. To balance the weights, we design a diagonal weight matrix $\mathbf{A} = diag(\mathbf{1}, \sqrt{|\mathcal{T}|}, \mathbf{1}, \mathbf{1}, \frac{\mathbf{1}}{\sqrt{|\mathcal{E}|}})$ based on the dimensions and the l_2 norm of the penalty, and then reformulate equation (37) as:

$$\mathcal{L}_{\sigma}(\{\boldsymbol{x}^{kv}\}, \boldsymbol{y}; \{\mathring{\boldsymbol{u}}^{kv}\}) = \sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}}\left\{\mathring{f}^{kv}(\boldsymbol{x}^{kv}) + \langle\mathring{\boldsymbol{u}}^{kv}, \mathbf{A}(\boldsymbol{x}^{kv} - \boldsymbol{y}^{kv})\rangle + \frac{\sigma}{2}||\mathbf{A}(\boldsymbol{x}^{kv} - \boldsymbol{y}^{kv})||^2\right\} + \mathcal{I}_{\mathcal{A}}(\boldsymbol{y}). \tag{47}$$

Then, the 2-block **ADMM** solution (in a scaled form) can be written as:

$$\begin{cases} \boldsymbol{x}_{i+1}^{kv} := \underset{\boldsymbol{x}^{kv}}{\operatorname{argmin}}\left\{\mathring{f}^{kv}(\boldsymbol{x}^{kv}) + \frac{\sigma}{2}||\mathbf{A}(\boldsymbol{x}^{kv} - \boldsymbol{y}_{i}^{kv} + \mathbf{A}^{-1}\hat{\boldsymbol{u}}_{i}^{kv})||^2\right\}, \text{ s.t. } \boldsymbol{x}^{kv} \in \mathring{\mathcal{X}}^{kv}, & (48)\\ \boldsymbol{y}_{i+1} := \underset{\boldsymbol{y}}{\operatorname{argmin}}\left\{\mathcal{I}_{\mathcal{A}}(\boldsymbol{y}) + \frac{\sigma}{2}\sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}}||\mathbf{A}(\boldsymbol{y}^{kv} - \boldsymbol{x}_{i+1}^{kv} - \mathbf{A}^{-1}\hat{\boldsymbol{u}}_{i}^{kv})||^2\right\}, & (49)\\ \hat{\boldsymbol{u}}_{i+1}^{kv} := \hat{\boldsymbol{u}}_{i}^{kv} + \mathbf{A}(\boldsymbol{x}_{i+1}^{kv} - \boldsymbol{y}_{i+1}^{kv}). & (50)\end{cases}$$

For above equations, we abuse notation $\boldsymbol{u}^{kv}$ and let $\boldsymbol{u}^{kv} = \mathbf{A}^{-1}\mathring{\boldsymbol{u}}^{kv}$ to simplify the formulations, equation (50) is transformed to the same form as equation (40). Based on the derivation in Appendix B, the solution for the new $\boldsymbol{y}$-update (49) can be also transformed to the same form as the solution (41). Finally, the only change is $\boldsymbol{x}$-update which solves equation (48) instead of (38).

Algorithm 6 *Distributed Resource Scheduling*

```
1: Initialize $\boldsymbol{y}_0^{kv} = 0, \boldsymbol{u}_0^{kv} = 0, \sigma = 0.2, i = 0$;
2: while True do
3:     Compute $\boldsymbol{x}_{i+1}^{kv}, \forall k, v$ in parallel by optimizing (48) (with $\boldsymbol{y}_i^{kv}, \boldsymbol{u}_i^{kv}$);
4:     Aggregate $\boldsymbol{x}_{i+1}^{kv}$ to update $\boldsymbol{y}_{i+1}^{kv}, \forall k, v$ in parallel via Equation (41) (with $\boldsymbol{x}_{i+1}^{kv}, \boldsymbol{u}_i^{kv}$);
5:     Update $\boldsymbol{u}_{i+1}^{kv}, \forall k, v$ in parallel via Equation (40) (with $\boldsymbol{x}_{i+1}^{kv}, \boldsymbol{y}_{i+1}^{kv}$);
6:     Compute primal and dual residuals ($\zeta_{i+1}^{primal}, \zeta_{i+1}^{dual}$) via Equations (51) and (52);
7:     if $\zeta_{i+1}^{primal} \leqslant \epsilon_{i+1}^{primal}$ & $\zeta_{i+1}^{dual} \leqslant \epsilon_{i+1}^{dual}$ then
8:         Return $\boldsymbol{x}_{i+1}^{kv}$;
9:     Update penalty parameter $\sigma$ via Equation (56);
10:    $i := i + 1$;
```

Correspondingly, the primal and dual residuals are rewritten as:

$$\begin{cases} \zeta_{i+1}^{primal} = ||\mathbf{A}(\boldsymbol{x}_{i+1}^{kv} - \boldsymbol{y}_{i+1}^{kv})||, & (51) \\ \zeta_{i+1}^{dual} = \sigma||\mathbf{A}^\mathsf{T}\mathbf{A}(\boldsymbol{y}_{i+1}^{kv} - \boldsymbol{y}_i^{kv})||. & (52) \end{cases}$$

And the feasibility tolerances are expressed are:

$$\begin{cases} \epsilon_{i+1}^{primal} = \epsilon^{abs}\sqrt{n} + \epsilon^{rel}\max\{||\mathbf{A}\boldsymbol{x}_{i+1}^{kv}||, ||\mathbf{A}\boldsymbol{y}_{i+1}^{kv}||\}, & (53) \\ \epsilon_{i+1}^{dual} = \epsilon^{abs}\sqrt{n} + \epsilon^{rel}\sigma||\mathbf{A}\mathbf{A}\boldsymbol{u}_{i+1}^{kv}||. & (54) \end{cases}$$

To sum up, our ADMM-based distributed resource scheduling is defined in Algorithm 6.

An advantage of ADMM is that the solving of the optimization problem is *distributed* among all or part of the edge computing nodes. Specifically, one edge node will play a role of management to distribute and coordinate the whole optimization process. Each edge node v will separately compute the optimizations for $\boldsymbol{x}_{i+1}^{kv}$, $\forall k$ ($\boldsymbol{x}$-update) in its local place, and each sub-task k could be also computed in parallel inside edge node v (leveraging the different computing cores or servers) based on the solution (see Equation (38)). After all the computation is done, the management node will collect and aggregate the intermediate solution $\boldsymbol{x}_{i+1}^{kv}, \forall k, v$ to generate information, e.g., $\overline{\boldsymbol{z}}_{i+1}^{k}$, ${}_{z}\overline{\boldsymbol{u}}_i^{k}$, $\overline{q}_{i+1}^{k}$, $\overline{\boldsymbol{r}}_{i+1}^{v}$, etc., (see Equation (41)), and also update the penalty parameter for the next iteration. Then, this information will be delivered to all edge nodes for continuing the next optimization round. During each iteration, the management node will check the convergence conditions to decide whether to finalize the whole tasks and report the final acceptable solution $\boldsymbol{x}_{i+1}^{kv}$, $\forall k, v$ based on the convergence condition (44).

5.5. Comment on convergence

To make the ADMM algorithm convergence fast, the penalty parameter (σ) can be properly tuned in each iteration of the solving procedure. In the ADMM update equations, a large value of σ gives a large penalty on violations of primal feasibility and hence tends to produce small primal residuals. Conversely, a small σ value tends to reduce the dual residual, but at the expense of reducing the primal feasibility and producing a larger primal residual. A general method is introduced in [40, 41] for balancing the variations of both primal and dual residuals, which is mainly designed for convex programming problems. However, it may show some instability and non-convergence properties in some application scenarios [42]. Here, we propose problem-specific modifications on this method for the σ updating strategy, which suites to the characteristics of our optimization model (i.e., mixed-integer, non-linear, non-convex programming). We first introduce the original strategy of [40, 41] as follows.

$$\sigma_{i+1} = \begin{cases} \sigma_i\omega^{incr}, & \text{if } ||\zeta_i^{primal}|| > \varphi||\zeta_i^{dual}||, \\ \sigma_i/\omega^{decr}, & \text{if } ||\zeta_i^{dual}|| > \varphi||\zeta_i^{primal}||, \\ \sigma_i, & \text{otherwise,} \end{cases} \quad (55)$$

where $\omega^{incr} > 1, \omega^{decr} > 1$ and $\varphi > 1$ are parameters. Typical values can be $\omega^{incr} = \omega^{decr} = 2$ and $\varphi = 10$. The idea behind the above strategy (55) is to try to keep the primal and dual residual norms within a small factor (φ) of one another as they both converge to zero.

Our proposed strategy is shown as follows:

$$\sigma_{i+1} = \begin{cases} \sigma_i \omega^{incr}, & \text{if } ||\zeta_i^{primal}|| > \varphi ||\zeta_i^{dual}|| \text{ or } \mathsf{stuck\_in\_local\_trap}, \\ \sigma_i, & \text{otherwise.} \end{cases} \tag{56}$$

where stuck_in_local_trap is to detect whether the ADMM algorithm is stuck in a local trap during the solving procedure. We define the local trap as a state when either of the primal and dual residuals keep unchanged for a certain interval, e.g., more than 5 iterations. Compared with the strategy in [40, 41], we also eliminate the decreasing statement of σ, which, in practice, is a cause of instability and non-convergence of the algorithm.

We will illustrate in the next section how these approaches permit to make the algorithm converge in practical network scenarios considered in our numerical evaluation.

6. Numerical Results

In this section we evaluate the performance of the proposed model, the *SFS* and *Greedy* heuristics, as well as the distributed (ADMM-based) resource allocation algorithm in terms of the profit of the operator, expressed as in ($\mathcal{P}.\mathcal{R}1$), the serving rate (the fraction of admitted requests) and the computation time to get the solution.

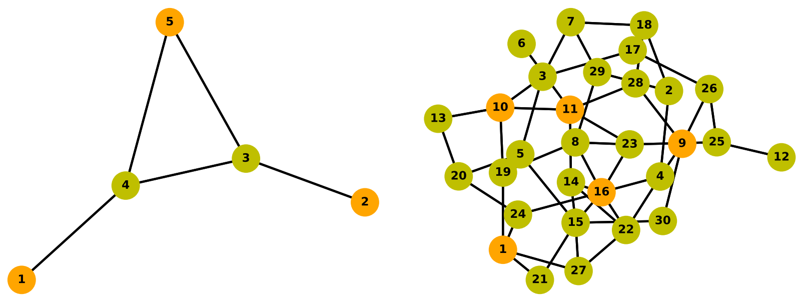
Consequently, the rest of this section is organized as follows: section 6.1 presents the network topologies we have considered in our numerical evaluation campaign; section 6.2 describes the setup for our experiments; finally, section 6.3 discusses the results obtained in different network scenarios.

6.1. Network Topologies

We evaluate our optimization approach using multiple network topologies, described hereafter, including several random graphs as well as a topology built on a real network scenario.

6.1.1. *Random graphs*

We first consider Erdös-Rényi random graphs [43], setting the desired 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 patch it with a simple strategy that connects isolated nodes to randomly sampled nodes (up to 10 nodes) in the graph. We generate several kinds of topologies with different numbers of nodes and edges, starting from simple ones to larger and more complex networks, as shown in Figure 4. The structural information for all topologies (including the one obtained in the real network scenario illustrated in the following) is reported in Table 4. All topology datasets are publicly available in our repository[1]. These topologies can be considered representative of various edge network configurations where multiple edge nodes are distributed in various ways over the territory.



(a) 5N5E

(b) 30N50E

Figure 4: Random network topologies: (a) 5 nodes and 5 edges; (b) 30 nodes and 50 edges. Ingress nodes for each graph are colored in orange.

[1] https://github.com/bnxng/Topo4Edge

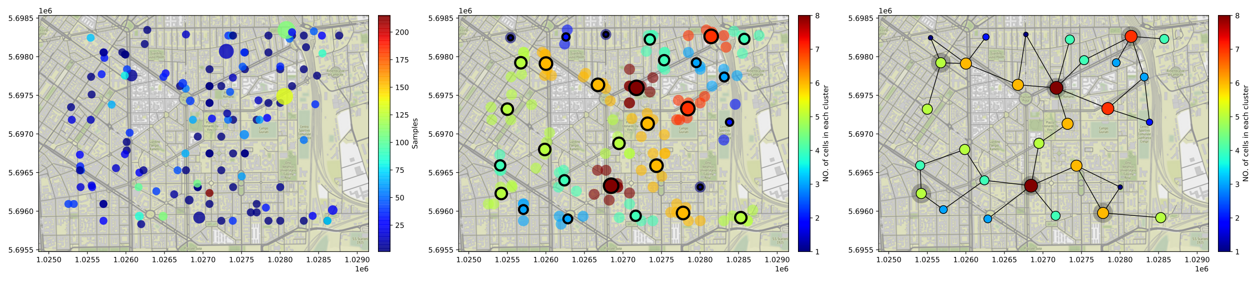

(a) Vodafone LTE cells (b) Cell clusters (c) Topology on clusters

Figure 5: Città Studi topology with 30 nodes, 35 edges and 6 ingress nodes (marked with gray shadow).

Table 4
Structural information of the topologies used in the experiments.

Topology	#Nodes	#Edges	#Ingress	Degree (Min, Max, Avg)	Diameter
5N5E	5	5	3	(1.0, 3.0, 2.0)	3
30N50E	30	50	5	(1.0, 7.0, 3.3)	5
CittàStudi	30	35	6	(1.0, 6.0, 2.3)	10

6.1.2. *A real network scenario*

We further consider a real network scenario, with the actual deployment of Base Stations collected from an open database, *OpenCellID*[2], which collects information of BSs from all over the world, including their positions. This topology was first introduced in [44], but in this paper we use it in a different context, solving the resource calendaring problem in a MEC context. 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 5(a)).

We then performed a clustering on such cells, as illustrated in Figure 5(b), obtaining 30 clusters.

Finally, we generated the network topology which, as in real mobile scenarios, has a fat tree-like shape with edge nodes connecting aggregation nodes, in the following way, starting from the cluster centroids:

- we connected any two nodes if their distance is lower than a given threshold (800 meters). By doing so, note that some "leaf"/edge nodes become connected to more than one *aggregation* node to increase redundancy and hence reliability of the final topology, as it happens in real networks. In Figures 5(b) and 5(c) the color (illustrated in the vertical bars) corresponds to the cluster size (number of cells contained in the cluster), with nodes aggregating more than 5 cells being regarded as the *aggregation* nodes;
- we determined the Minimum Spanning Tree of the geometric graph weighted by the distance and cluster size, while preserving some redundant links mentioned above.

The resulting topology is illustrated in Figure 5(c); the average node degree resulting from the above procedure is 2.33. In such topology, edge servers can be installed in all nodes.

6.2. Experimental Setup

We implemented our model and heuristics using SCIP (Solving Constraint Integer Programs)[3], 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 results illustrated in the following figures are obtained by averaging over 50 instances, with 97% narrow confidence intervals.

We uniformly extract, at random, source nodes as well as the starting/ending times and duration, and the revenue gained by the operator in serving each request, in the [100, 300] range. We further generate random request rates on the ingress edge nodes of the network topologies (see Figures 4(a), 4(b) and 5(c)) according to a Gaussian distribution $N(\lambda^k, \sigma^2)$, where λ^k is uniformly selected in the 30 to 50 Gbps range and $\sigma = 0.5$. We further consider more complex

[2]`https://www.opencellid.org/`
[3]`http://scip.zib.de`

Table 5
Parameters setting - initial (reference) request data (for the case of high incoming request load).

Request ID	$\alpha^k(slot)$	$\beta^k(slot)$	$d^k(slot)$	$\lambda^k(Gbps)$	$\mu^k(€)$
A1	3	12	5	50	200
A2	1	8	3	40	100
A3	5	20	8	45	200
A4	3	16	8	35	300
A5	5	18	6	55	250
A6	7	22	9	30	150

Table 6
Network scenarios considered in the experiments.

Scenario	Topology	Requests on Ingress	Random Split	No. of Requests
5N5E3R	*5N5E*	A1-A3	-	3
30N50E30R	*30N50E*	A1-A5	6	30
CittàStudi6R	*CittàStudi*	A1-A6	-	6
CittàStudi30R	*CittàStudi*	A1-A6	5	30

scenarios by randomly splitting the "heavy" requests on each ingress node to spawn a variety of "small" different requests; specifically, for topology *30N50E* (see Table 4), we generate one request on each of the five ingress nodes, then split each request into 6 parts to create a network scenario *30N50E30R* having a total of 30 requests; for topology *Città Studi*, in the same way, we first create a scenario *CittàStudi6R* having 6 requests where each ingress node holds one, then each request is split into 5 parts to generate a scenario *CittàStudi30R* having a total of 30 requests. For the sake of simplicity, we assume that all links have the same bandwidth ($B_e = 30$ Gbps) and nodes have the same computation capacity ($D_v = 30$ Giga cycles/s) and storage capacity ($S_v = 40$ GB). The costs of using one unit of these three resources, ψ_e, θ_v, and ϕ_v, are all set to 0.01. Finally, we set the processing density $\eta^k = 1$ and the storage requirement $m^k = 10$ for all requests.

In Table 5, we provide a summary of the reference values we define for the main parameters related to requests. Such values are representative of a scenario with a high load of requests relative to the limited computation. Our request rates result from the aggregation of requests generated by multiple users connected at a given ingress node. More specifically, the values of request rate λ^k are designed to cover several different scenarios, i.e., *mice*, *normal* and *elephant* request load. We select rate values and requests duration (starting and ending times) which are typical of a 5G usage scenario (eMBB, URLLC, and mMTC) [45]. For instance, a request which has a very short computation duration and a small rate could represent an URLLC use case, and a mission critical application, while a request with a higher computation duration and a high rate could represent an eMBB use case and an augmented reality service. Specifically, the white paper [45] describes a COSMOTE (a provider of Edge Computing infrastructure) 5G testbed with an Openstack-based multi-cloud infrastructure interconnected with 10 Gbps fiber/copper links. In case of smart metropolitan areas, the network capabilities and requirements of MEC hosts are investigated in a survey [46], which shows that every MEC host can use, on average, at least 62.5 Gbps in downlink and 10.41 Gbps in uplink. In [47], the authors study the network requirements to realize a use case of MEC-based AR assisted remote surgery, which requires a bandwidth of at least 30 Gbps. In this work we are using parameters for our demands that are in the range of those indicated in the above works, e.g., in our settings, the request rates at ingress nodes vary from 30 to 50 Gbps, and the link bandwidth is set to 30 Gbps. In Table 5, almost all requests cannot be served using only the resources (computation capacity) at their respective ingress nodes. Note that our proposed model and heuristics are general, and can be applied to optimize resource allocation in all network scenarios with any parameters setting.

Table 6 summarizes for each network scenario we considered in our numerical evaluation the network topology used, the users requests offered to the network (following Table 5 definitions), how they are split and the total number of requests.

Parameters' setting has an impact on the performance of the proposed heuristic algorithms, both in terms of quality of the obtained solution and execution time. Hereafter, we consider the *Greedy* algorithm as an example. Table 7

Table 7
Effect of the w_{greedy} parameter on the profit obtained by the *Greedy* approach and corresponding computing time under three scenarios with computation capacity D_v scaled by 0.6, $\forall v$.

Scenario	w_{greedy}		
(D_v scaled by 0.6, $\forall v$)	0.4	0.6	0.8
CittàStudi6R	645.74 (49 s)	**838.64 (30 s)**	321.10 (7 s)
CittàStudi30R	**850.26 (297 s)**	837.12 (189 s)	822.73 (125 s)
30N50E30R	740.15 (524 s)	**747.96 (665 s)**	747.59 (644 s)

illustrates the effect of w_{greedy} on the profit value and on the corresponding computing time of the *Greedy* approach under three demanding scenarios (with limited computation capacity, obtained scaling the D_v parameter by a factor 0.6, $\forall v$). The w_{greedy} parameter permits to strike a balance between the obtained performance and the computing time. In fact, *Greedy* obtains a higher profit value at $w_{greedy} = 0.6$ in almost all scenarios except *CittàStudi30R*, where it has slightly lower profit value than that of point 0.4, but with a much shorter computing time. In scenario *CittàStudi6R*, *Greedy* obtains much better profit at point 0.6 than that obtained at points 0.4 an 0.8 with an increase of a factor 1.3 and 2.6, respectively. Finally, the value of w_{greedy} used in the following subsections is set to 0.6.

6.3. Discussion of Results

In the following, we first compare the results obtained from both the exact model and the heuristics, including our ADMM-based approach, for a small network scenario: the *5N5E* network with 3 requests (represented by *5N5E3R*) in section 6.3.1. Then, in section 6.3.2, we analyze the effect of different parameters on the solution obtained by the two heuristics *SFS* and *Greedy* for two large network scenarios, i.e., one random topology *30N50E* and one real network scenario *Città Studi* with 30 requests (represented by *30N50E30R* and *CittàStudi30R*, respectively). For this latter topology, to model different levels of demand aggregation, we considered a further scenario (denoted by *CittàStudi6R*) with 6 requests that are the aggregations of the above 30 requests at the 6 ingress nodes, while maintaining the same total request rates.

6.3.1. Comparison of the exact model and heuristics

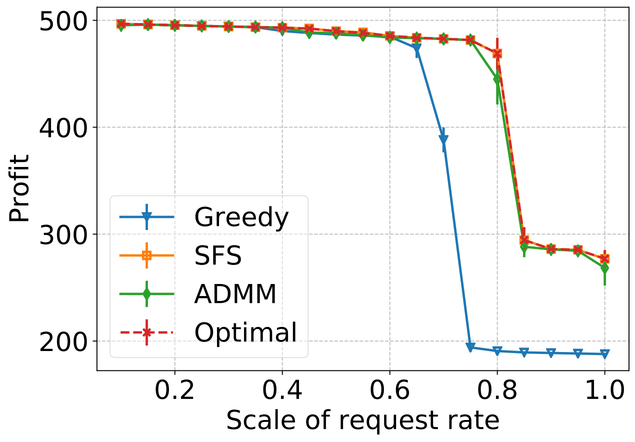


Figure 6: Profit against request rate (λ^k) for scenario *5N5E3R*.

The *5N5E3R* topology allows us to compare to the optimal solution the solutions obtained from the heuristics (*SFS* and *Greedy*) as well as the distributed algorithm *ADMM*. In fact, the exact model ($\mathcal{P}.\mathcal{R}1$) could be solved in a reasonable time only in the small topology (*5N5E3R*). Figure 6 plots the profit versus the request rate λ^k keeping the revenue μ^k fixed, where *Optimal* represents the result obtained by solving the exact model ($\mathcal{P}.\mathcal{R}1$). The decreasing trend of the profit for all the approaches, when increasing λ^k, is due to the fact that more resources are needed, hence the cost incurred by the operator increases while the revenue is fixed. This results into a profit decrease. The profit drops to

Table 8
Comparison of computing time (seconds).

Scenario	*Optimal*	*SFS*	*Greedy*	*ADMM*
5N5E3R	62	1.3	1.9	59
30N50E30R	-	1096	822	6676|890
CittàStudi6R	-	19	36	365
CittàStudi30R	-	362	325	6631|884

around 300 for *SFS*, *ADMM* and *Optimal*, while around 200 for *Greedy*. The curves show a step-wise pattern due to the combinatorial expression of the profit in the objective function of the optimization ($\mathcal{P}.\mathcal{R}1$) and the only 3 requests contained in instance *5N5E3R*. Both *ADMM* and *SFS* exhibit excellent performance since their curves practically overlap that of *Optimal*, while *Greedy* shows lower performance. In the small network scenario, *5N5E3R*, *ADMM* and *SFS* can obtain good results mainly due to two reasons: i) the solution space of the optimization model for *5N5E3R* is relatively small and simple, ii) the algorithms can capture the main issues of the problem: taking into account requests' priority and overlapping, namely in deciding their admission into the system, as well as performing an effective exploration of candidate computing nodes, etc, which all influence significantly the quality of the obtained solution. However, we can expect that in larger scenarios, a certain gap exists with respect to the optimum. As for the *Greedy* algorithm, its lower performance is mainly due to its differences with respect to *SFS*. Specifically, the *Greedy* approach adopts different strategies to prioritize requests and to search candidate nodes for processing requests. Besides, it does not consider requests' overlap and the exploration of solutions in case of infeasibility. Therefore, it may show lower performance.

As for the computing time, Table 8 summarizes this performance figure for our proposed algorithm in the considered network scenarios; for example, in the *5N5E3R* scenario, *Optimal* has an average computing time of 62 s, *ADMM* of 59 s, while *Greedy* takes 1.9 s and *SFS* just 1.3 s. A detailed discussion on the results of large network scenarios is provided in the following subsections. As for the two *ADMM* values related to topologies *30N50E30R* and *CittàStudi30R*, the first is the computing time measured in the simulation server used in our measurement campaign, with 20 CPU Cores, and the second is the estimated computing time for a real edge computing network where the computation of subproblems can be fully distributed. More details are provided below when discussing the results in Table 9.

6.3.2. Analysis of SFS and Greedy heuristics' results for large networks

In the following, we illustrate the objective function value, in terms of *profit*, as a function of different parameters for the two network scenarios *30N50E30R* and *CittàStudi30R* (Figures 7-11) and *serving rate*, which is plotted as a function of the request rate and revenue (λ^k, μ^k) (see Figure 9).

Effect of the request rate (λ^k)**:** Figures 7(a) and 7(b) report the *profit* as the function of the request rate λ^k, scaled from 0.5 to 2.0 with respect to their initial values (see Table 5), for scenarios *30N50E30R* and *CittàStudi30R*, respectively. As λ^k increases, the profits for all approaches in the two scenarios decrease, since the revenue from serving each request is fixed while the system cost for serving the growing requests increases. In Figures 7(a) and 7(b), the *SFS* curves show a similar trend as λ^k increases, while *Greedy* curves follow a slightly different pattern, specifically, the profit in *30N50E30R* decreases smoothly as λ^k increases, while in *CittàStudi30R*, the profit rapidly decreases after the scaling point 1.4. Finally, *SFS* performs better than *Greedy* in both scenarios with average gaps up to 8% and 9%, respectively.

Effect of the request rate and revenue (λ^k, μ^k)**:** Figures 8(a) and 8(b) illustrate the profit variation versus the request rate and revenue for scenarios *30N50E30R* and *CittàStudi30R*, respectively. Values of λ^k and $\mu^k,\ k \in \mathcal{K}$ are both scaled, at the same time, from 0.5 to 2.0 with respect to their initial values. Such scaling implicitly indicates that serving each request provides a revenue proportional to its arrival rate. As (λ^k, μ^k) increase, the *profits* (see Figures 8(a) and 8(b)) for *SFS* increase, showing an opposite trend compared with Figures 7(a) and 7(b); the network operator, in fact, is able to select and admit the requests which can cover the system cost and provide, at the same time, higher profit. In Figure 8(b), when the scale is larger than 1.7, *Greedy* shows a slight decreasing trend, since it fails to find a good solution to balance the cost and profit.

Figures 9(a) and 9(b) illustrate the variations of serving rate versus the request rate and revenue for scenarios *30N50E30R* and *CittàStudi30R*, respectively. In Figure 9(a), when the request rate is low, all user requests can be served; when it increases, specifically after the point around 1.2, the *serving rate* of *SFS* decreases since the system can accommodate less requests, which become more demanding, hence costlier in terms of required resources. In scenario *CittàStudi30R* (see Figure 9(b)), when the scale factor is lower than 1.25, the serving rates for both approaches slowly decrease as the request rates increase; after that, the decrease for *Greedy* becomes rapid while for *SFS* it is slower. Finally, *SFS* exhibits better performance compared to *Greedy*, with gaps up to 18% for the profit and 20% for the serving rate.

Effect of the link capacity B_e: The variation of the profit as a function of the link capacity B_e (scaled from 0.1 to 1.2 with respect to its initial value), is illustrated in Figures 10(a) and 10(b) for scenarios *30N50E30R* and *CittàStudi30R*, and they show a very similar trend. When B_e increases, the profit increases for both *SFS* and *Greedy* in the two scenarios. Both of them increase fast before the scaling point 0.75, reflecting the positive effect of the available link capacity on the profit. Additionally, *SFS* performs better than *Greedy* with clear gaps: up to 77%. For larger values of the available link capacity, there are naturally enough resources to satisfy the requests' requirements; *SFS* and *Greedy* hence perform similarly and converge to specific values, and the gap between them also decreases.

Effect of the computation capacity D_v: Figures 11(a) and 11(b) show the variations of the profit against the edge node computation capacity D_v, scaled with respect to its initial value from 0.5 to 1.5, in scenarios *30N50E30R* and *CittàStudi30R*. When D_v increases, the profit first increases rapidly and then converges to a specific plateau value for all approaches. Note that, for all allocation algorithms, the increase in terms of achieved profit can be up to 300, while the increase of the serving rate is around 0.4. These trends reflect the strong effect of the available computation capacity on the profit and serving rate. Additionally, *SFS* performs better than *Greedy* with clear gaps (up to 13% for the profit). With the increase of computation capacity, the performance gap between *SFS* and *Greedy* decreases since the utilization of enhanced algorithms is less critical to perform a good resource allocation, when resources are abundant. In both network scenarios, *SFS* allows the operator to achieve higher profit, which stabilizes when the scale of computation capacity is above the 0.9 value, while *Greedy* converges after the computation capacity is scaled up to around 1.1 for *30N50E30R*, and around 0.9 for *CittàStudi30R*.

Finally, in Table 8, *SFS* exhibits an average computing time of 1096 s in scenario *30N50E30R* and of 362 s in scenario *CittàStudi30R*, confirming its efficiency in computing good solutions in a short time. The *Greedy* approach needs less computation time, on average 822 s in *30N50E30R* and 325 s in *CittàStudi30R*, to obtain a solution, at the cost of higher performance gaps with respect to the *SFS* heuristic. Besides, compared with *30N50E30R*, *CittàStudi30R* requires slightly less computing time to obtain the solutions since its topology has a relatively smaller size and a *fat-tree* structure. Generally, when the network scenario is small (e.g., *5N5E3R*, *CittáStudi6R*), *SFS* can be slightly faster than *Greedy*. In this case, we observe that the quality of the heuristic trial solutions for the subproblems influences the efficiency of the optimization solvers. *SFS* provides better solutions, compared with *Greedy*. On the other hand, for larger networks, the exploration of *SFS* becomes heavier, which consumes a larger amount of time. Since the *Greedy* algorithm does not have this overhead, in such scenarios it runs faster than *SFS*. For *ADMM*, a detailed discussion on both the measured and estimated computing times under the *30N50E30R* and *CittàStudi30R* scenarios is provided below in Section 6.3.3.

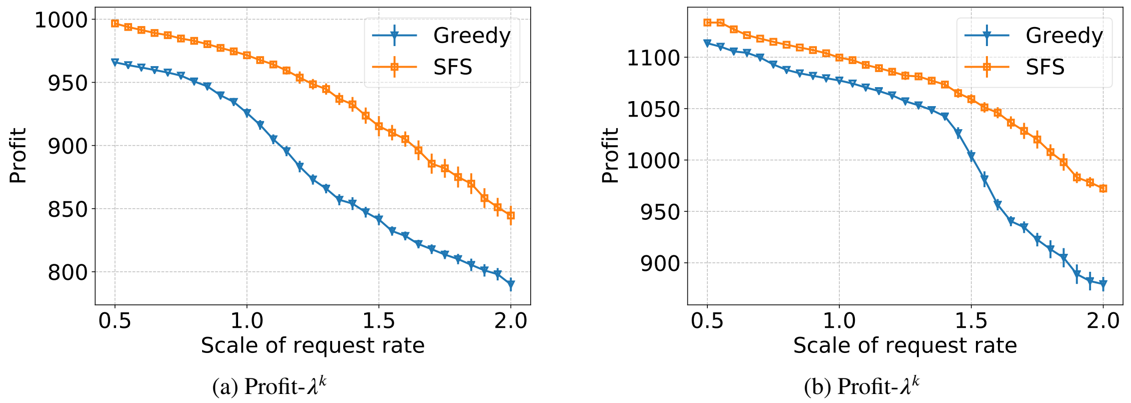


Figure 7: Profit against scaling parameter λ^k for scenarios (a) *30N50E30R*, (b) *CittàStudi30R*.

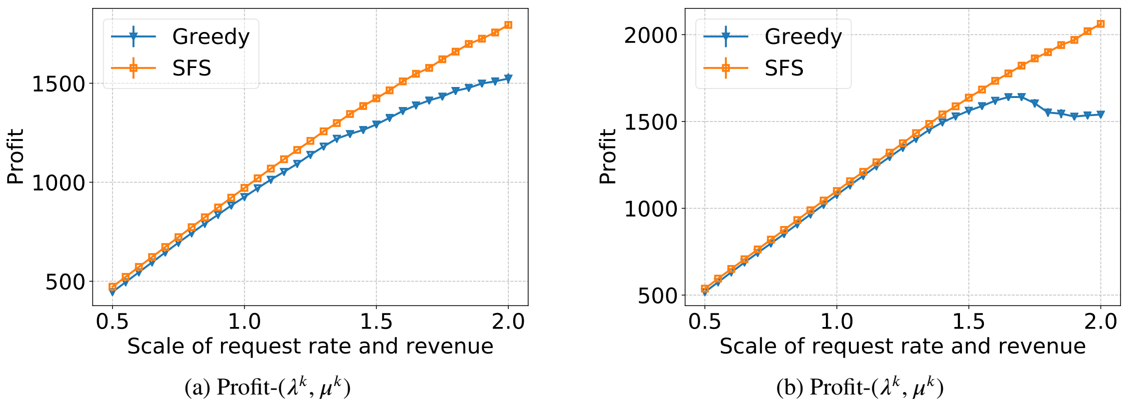


Figure 8: Profit against scaling parameters (λ^k, μ^k) for scenarios (a) *30N50E30R*, (b) *CittàStudi30R*.

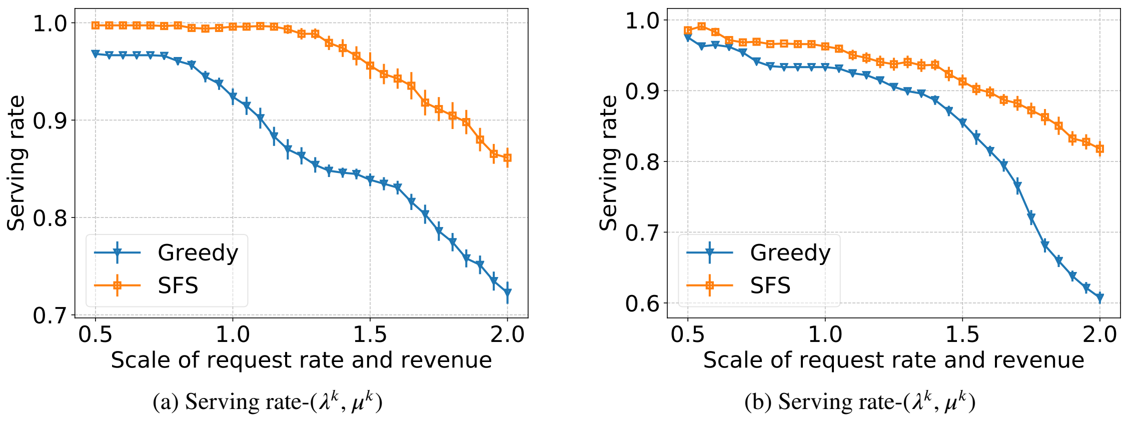


Figure 9: Serving rate against scaling parameters (λ^k, μ^k) for scenarios (a) *30N50E30R*, (b) *CittàStudi30R*.

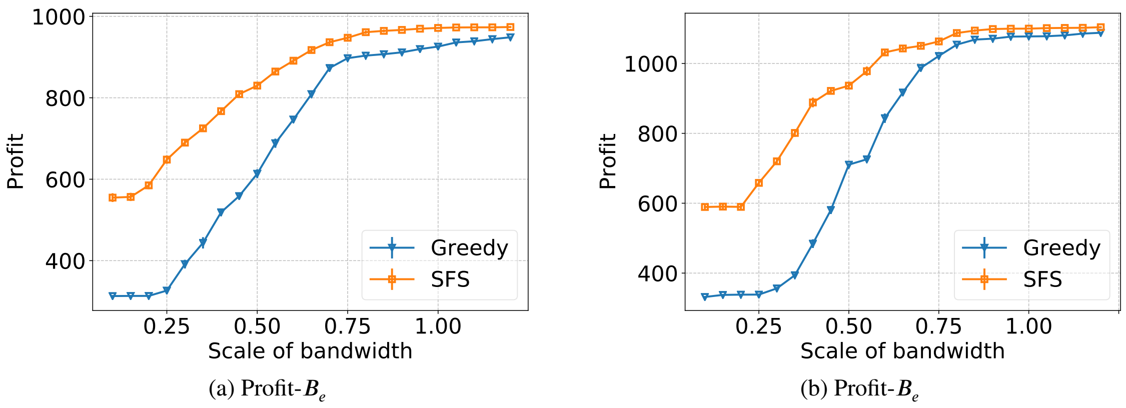


Figure 10: Profit against scaling parameter B_e for scenarios (a) *30N50E30R*, (b) *CittàStudi30R*.

6.3.3. Analysis of ADMM results

To measure the performance of the distributed, ADMM-based algorithm described in Section 5, we run experiments comparing the behavior of ADMM, *SFS* and *Greedy* in the *CittàStudi6R* network scenario with 6 requests. Figures 12(a), 12(b) and 12(c) illustrate the variation of the profit as a function of scaling parameters λ^k, B_e, and D_v, respectively. The curves in the sub-figures have trends similar to those in Fig. 7, 10 and 11, respectively, thus confirming the effects of these parameters on the solutions obtained in different network scenarios. The curves also show a

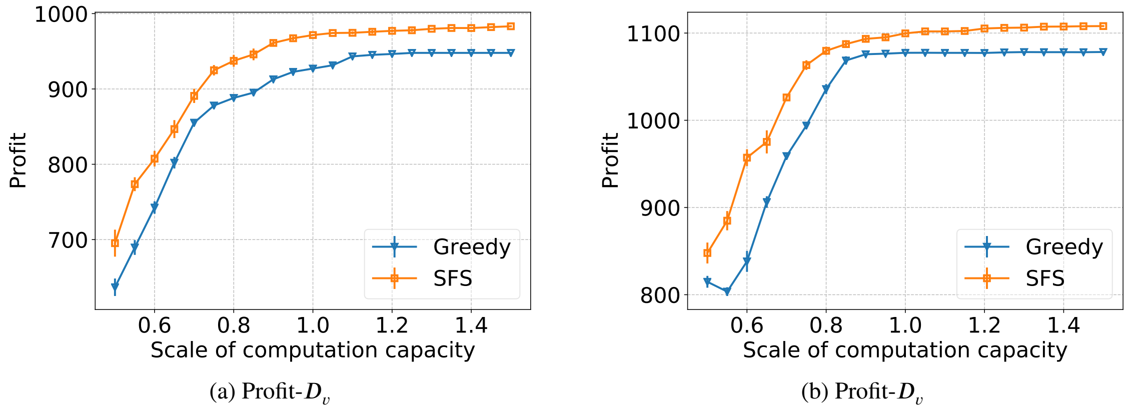


Figure 11: Profit against scaling parameter D_v for scenarios (a) *30N50E30R*, (b) *CittàStudi30R*.

step-wise pattern (like Figure 6) due to the same reasons highlighted before. Note that the curves practically overlap, except for very high traffic requests/scaling factors (see Fig. 12(a), particularly with scaling factors below 1.7); the same behavior can be observed in Fig. 12(b), especially for scaling values larger than 0.9, as well as in Fig. 12(c) for scaling factors above 0.8). This indicates that our proposed distributed algorithm achieves practically the same performance as the centralized algorithm (SFS) in a large set of real network settings.

As for the computing time, *SFS* requires, on average, 19 s, *Greedy* takes 36 s, and *ADMM* takes 365 s, with an average of 143 iterations. The main reason for the higher computing time of *ADMM* is due to the number of iterations needed for the convergence, the computing time of each subproblem and also the parallelization of the solving process for all subproblems in the environment. For instance, regarding topology *CittàStudi6R*, the maximum number of subproblems to be computed in parallel for each iteration of *ADMM* is $|\mathcal{K}||\mathcal{V}| = 180$ and each subproblem takes around $1 \sim 2$ seconds. To solve the optimization in a reasonable time, we limit the number of neighbor edge nodes that can be explored to 5 nodes, which reduces the number of the subproblems to $5|\mathcal{K}| = 30$. The disadvantage is that it degrades the performance of *ADMM*. In each iteration of *ADMM*, these subproblems are solved on the simulation server which has 20 (< 30 subproblems) cores. As mentioned above, in this case, *ADMM* takes around 365 s with an average of 143 iterations. In a real edge network environment, all subproblems in each iteration can be solved distributedly on the edge nodes, and as a result both the performance and the computing time can be certainly improved. For a larger network scenario, *ADMM* has higher potential compared to *SFS* and *Greedy* w.r.t. the performance and computing time, since for *ADMM* these are mainly influenced by the solving process of each subproblem ($kv \in \mathcal{K} \times \mathcal{V}$) whose complexity only depends on $|\mathcal{T}|$, i.e., the number of slots considered in the time horizon, while for both *SFS* and *Greedy* algorithms it depends on the complexity of the original problem which increases exponentially w.r.t. the problem scale.

Table 9 compares the profit value obtained and the corresponding computing time of different approaches in network scenarios *CittàStudi30R* and *30N50E30R*. Three scaling points for the request rate $\lambda^k, \forall k$ are selected in each scenario. Regarding the profit value, *ADMM* performs better than *SFS* and *Greedy* except at scaling point 0.5 in the *30N50E30R* scenario, where the profit obtained by *ADMM* is slightly lower than that achieved by *SFS*. For the computing time, the Table reports for *ADMM* two values: one is the computing time measured in a simulation server with 20 CPU cores (the one we used in our measurement campaign), and the other is the estimated computing time for a real edge computing network where the computation of subproblems can be fully distributed. We estimate the value based on the following analysis. In the experiments, for requests from different ingress nodes, we make the algorithms (*SFS*, *Greedy*, *ADMM*) explore the 5 nearest neighbor nodes for each ingress node to limit the exploration space and accelerate the process assuming their total computation capacity is sufficient. Therefore, for *ADMM*, the number of parallelized subproblems is equal to $|\mathcal{K}||\mathcal{V}_{nb}| = 30 * 5 = 150$, and the theoretical computing time in real networks would be $20/150 = 1/7.5$ times lower than that obtained with the simulation server; for instance, in the *CittàStudi30R* scenario at point 0.5, the estimated computing time is $6676/7.5 = 890s$. Compared with *SFS* and *Greedy*, the estimated computing time of *ADMM* is larger in scenario *CittàStudi30R* due to the overhead of *ADMM* caused by the many iterations needed for a convergence to the solution. As the problem scale increases, the overhead of *ADMM* becomes negligible compared to the total computing time. For instance, in scenario *30N50E30R*, the estimated computing time of *ADMM* is almost at the same level as in scenario *CittàStudi30R*, it is also close to the computing time of *SFS*, and

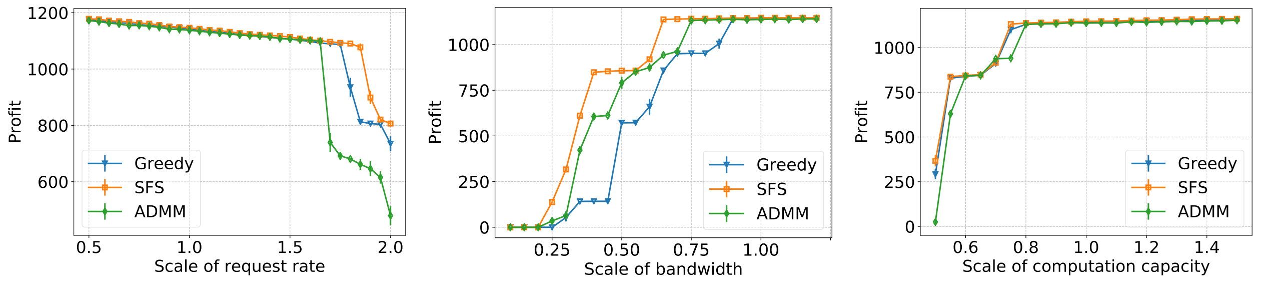
Table 9
Comparison among *ADMM*, *SFS* and *Greedy* in large network scenarios w.r.t. profit and computing time.

Approach	*CittàStudi30R* (scaling $\lambda^k, \forall k$)			*30N50E30R* (scaling $\lambda^k, \forall k$)		
	0.5	1.0	1.5	0.5	1.0	1.5
Greedy	1113, 341 s	1079, 313 s	1013, 279 s	966, 885 s	906, 815 s	851, 922 s
SFS	1137, 352 s	1104, 375 s	1081, 343 s	**999**, 875 s	970.6, 1032 s	881, 1126 s
ADMM	**1142**, 6676|890 s	**1116**, 6536|871 s	**1092**, 7907|1054 s	980, 6631|884 s	**970.8**, 7866|1049 s	**951**, 8144|1086 s

even lower than it at point 1.5. For *SFS* and *Greedy*, due to the higher complexity, the computing time increases of about 2.8 times in scenario *30N50E30R*.

Another aspect to be considered is that the performance of *ADMM* depends on the initial penalty parameter σ and the updating strategy (see Algorithm 6), which can be further tuned to achieve the best performance. In this work, we use empirical and intuitive settings for *ADMM*, for simplicity, and at the same time, to demonstrate the potential of a distributed algorithm applied in the resource scheduling for edge computing networks. A rigorous fine tuning will be the subject of future work.

Finally, we would like to emphasize that a key advantage of *ADMM* is that the optimization for resource scheduling can be solved distributedly on all edge computing nodes, while providing a very good solution for the operator. This feature of *ADMM* is very important: it allows the operator to alleviate the problems deriving from a single point of failure and to obtain a good scheduling solution in an environment where, in several practical situations, only a distributed scheme can be applied for optimizing the resource allocation and scheduling.



(a) Profit-λ^k (b) Profit-$\boldsymbol{B}_e$ (c) Profit-$\boldsymbol{D}_v$

Figure 12: Profit against scaling parameters λ^k, $\boldsymbol{B}_e$, and $\boldsymbol{D}_v$ for scenarios *CittàStudi6R*.

7. Conclusion

In this paper we formulated and solved the resource calendaring problem in mobile networks equipped with Mobile Edge Computing capabilities. Specifically, we first proposed an exact optimization model and an effective heuristic able to obtain a near-optimal solution in all the considered, real-size network scenarios.

We further proposed a distributed resource allocation algorithm, based on the ADMM method, that we extended using a *weighting* approach especially tailored to our problem, which allows each node to take local decisions coordinating its actions with the other nodes, converging reasonably fast to near-optimal solutions, as we illustrated in our numerical evaluation which includes both random geometric graphs and realistic mobile network topologies obtained from actual cell positions.

The decisions we optimized include admission control for the user requests offered to the network, their calendaring (scheduling) and bandwidth constrained routing, as well as the determination of which nodes provide the required computation and storage capacity. Calendaring, in particular, permits to exploit the intrinsic flexibility in the services demanded by different users, whose starting time can be shifted without penalizing the utility perceived by the user while, at the same time, permitting a better resource utilization in the network.

Other future research directions include more experiments on the ADMM approach, considering larger networks and fine tuning the initial penalty parameter and the updating strategy, as well as the implementation of the approach on a real network testbed. It would further be interesting to devise and numerically analyze other objective functions with a more sophisticated pricing model (e.g., with a fixed and a variable/proportional part according to the requests types and requirements [48]) for edge computing services. Another interesting point that is worth investigating is to extend the model capturing errors and subsequent retransmissions in the processing problem, which has an impact on the experienced latency and deadlines of users' requests.

Acknowledgments

We are very grateful to the anonymous reviewers who have significantly contributed to the improvement of this paper. This research was supported by the H2020-MSCA-ITN-2016 SPOTLIGHT under grant agreement number 722788 and the H2020-ICT-2020-1 PIACERE under grant agreement number 101000162.

References

[1] Salvatore D'Oro, Leonardo Bonati, Francesco Restuccia, Michele Polese, Michele Zorzi, and Tommaso Melodia. Sl-EDGE: Network Slicing at the Edge. In *ACM Mobihoc: International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 1–10, 2020.

[2] Chen-Feng Liu, Mehdi Bennis, Mérouane Debbah, and H Vincent Poor. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Transactions on Communications*, 67(6):4132–4150, 2019.

[3] Mohammed S Elbamby, Mehdi Bennis, Walid Saad, Matti Latva-Aho, and Choong Seon Hong. Proactive edge computing in fog networks with latency and reliability guarantees. *EURASIP Journal on Wireless Communications and Networking*, (209), 2018.

[4] Pengfei Wang, Zijie Zheng, Boya Di, and Lingyang Song. Joint Task Assignment and Resource Allocation in the Heterogeneous Multi-Layer Mobile Edge Computing Networks. In *IEEE GLOBECOM*, 2019.

[5] Jianan Zhang, Abhishek Sinha, Jaime Llorca, Antonia Tulino, and Eytan Modiano. Optimal control of distributed computing networks with mixed-cast traffic flows. In *IEEE INFOCOM*, pages 1880–1888, 2018.

[6] Maxime Dufour, Stefano Paris, Jérémie Leguay, and Moez Draief. Online Bandwidth Calendaring: On-the-fly admission, scheduling, and path computation. In *IEEE ICC*, 2017.

[7] Lin Wang, Fa Zhang, Kai Zheng, Athanasios V Vasilakos, Shaolei Ren, and Zhiyong Liu. Energy-efficient flow scheduling and routing with hard deadlines in data center networks. In *IEEE 34th International Conference on Distributed Computing Systems*, pages 248–257, 2014.

[8] Haisheng Tan, Zhenhua Han, Xiang-Yang Li, and Francis CM Lau. Online job dispatching and scheduling in edge-clouds. In *IEEE INFOCOM*, pages 1–9, 2017.

[9] Jiaying Meng, Haisheng Tan, Xiang-Yang Li, Zhenhua Han, and Bojie Li. Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1270–1286, 2020.

[10] Vajiheh Farhadi, Fidan Mehmeti, Ting He, Tom La Porta, Hana Khamfroush, Shiqiang Wang, and Kevin S Chan. Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds. In *IEEE INFOCOM*, 2019.

[11] Bin Xiang, Jocelyne Elias, Fabio Martignon, and Elisabetta Di Nitto. Joint network slicing and mobile edge computing in 5G networks. In *IEEE ICC*, 2019.

[12] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[13] Bin Xiang, Jocelyne Elias, Fabio Martignon, and Elisabetta Di Nitto. Resource Calendaring for Mobile Edge Computing in 5G Networks. In *IEEE ICC*, 2021.

[14] Yiwei Thomas Hou, Yi Shi, and Hanif D Sherali. Optimal spectrum sharing for multi-hop software defined radio networks. In *IEEE INFOCOM*, pages 1–9, 2007.

[15] Jocelyne Elias, Stefano Paris, and Marwan Krunz. Cross-Technology Interference Mitigation in Body Area Networks: An Optimization Approach. *IEEE Transactions on Vehicular Technology*, 64(9):4144–4157, 2014.

[16] Zaid Allybokus, Konstantin Avrachenkov, Jérémie Leguay, and Lorenzo Maggi. Multi-path alpha-fair resource allocation at scale in distributed software-defined networks. *IEEE Journal on Selected Areas in Communications*, 36(12):2655–2666, 2018.

[17] Lazaros Gkatzikis, Stefano Paris, Ioannis Steiakogiannakis, and Symeon Chouvardas. Bandwidth calendaring: dynamic services scheduling over software defined networks. In *IEEE ICC*, 2016.

[18] Mira Morcos, Jocelyne Elias, Fabio Martignon, Tijani Chahed, and Lin Chen. On efficient radio resource calendaring in cloud radio access network. *Computer Networks*, 162(106862), 2019.

[19] Qiang Liu and Tao Han. Direct: Distributed cross-domain resource orchestration in cellular edge computing. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 181–190, 2019.

[20] Zhenyu Zhou, Junhao Feng, Zheng Chang, and Xuemin Shen. Energy-efficient edge computing service provisioning for vehicular networks: A consensus admm approach. *IEEE Transactions on Vehicular Technology*, 68(5):5087–5099, 2019.

[21] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM*, volume 43, pages 15–26, 2013.

[22] Davide Andreoletti, Sebastian Troia, Francesco Musumeci, Silvia Giordano, Guido Maier, and Massimo Tornatore. Network traffic prediction based on diffusion convolutional recurrent neural networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 246–251. IEEE, 2019.

[23] Xiaofeng Cao, Yuhua Zhong, Yun Zhou, Jiang Wang, Cheng Zhu, and Weiming Zhang. Interactive temporal recurrent convolution network for traffic prediction in data centers. *IEEE Access*, 6:5276–5289, 2017.

[24] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 515–526, 2014.

[25] Subhash Suri, Marcel Waldvogel, Daniel Bauer, and Priyank Ramesh Warkhede. Profile-based routing and traffic engineering. *Computer communications*, 26(4):351–365, 2003.

[26] Jocelyne Elias, Fabio Martignon, Antonio Capone, and Guy Pujolle. A new approach to dynamic bandwidth allocation in quality of service networks: performance and bounds. *Computer Networks*, 51(10):2833–2853, 2007.

[27] Xiao Ma, Shangguang Wang, Shan Zhang, Peng Yang, Chuang Lin, and Xuemin Sherman Shen. Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing. *IEEE Transactions on Cloud Computing*, 2019.

[28] Lixing Chen, Sheng Zhou, and Jie Xu. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Transactions on Networking*, 26(4):1619–1632, 2018.

[29] Phuong Luong, François Gagnon, Charles Despins, and Le-Nam Tran. Joint virtual computing and radio resource allocation in limited fronthaul green C-RANs. *IEEE Transactions on Wireless Communications*, 17(4):2602–2617, 2018.

[30] Jeongho Kwak, Yeongjin Kim, Joohyun Lee, and Song Chong. DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE Journal on Selected Areas in Communications*, 33(12):2510–2523, 2015.

[31] Yipei Niu, Bin Luo, Fangming Liu, Jiangchuan Liu, and Bo Li. When hybrid cloud meets flash crowd: Towards cost-effective service provisioning. In *IEEE INFOCOM*, pages 1044–1052, 2015.

[32] Chih-Ping Li, Jing Jiang, Wanshi Chen, Tingfang Ji, and John Smee. 5G ultra-reliable and low-latency systems design. In *European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, 2017.

[33] Liqing Liu, Zheng Chang, Xijuan Guo, Shiwen Mao, and Tapani Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5(1):283–294, 2017.

[34] Jianhua Tang, Wee Peng Tay, Tony QS Quek, and Ben Liang. System cost minimization in cloud RAN with limited fronthaul capacity. *IEEE Transactions on Wireless Communications*, 16(5):3371–3384, 2017.

[35] Binnan Zhuang, Dongning Guo, and Michael L Honig. Energy-efficient cell activation, user association, and spectrum allocation in heterogeneous networks. *IEEE Journal on Selected Areas in Communications*, 34(4):823–831, 2016.

[36] Ravindran Kannan and Clyde L Monma. On the computational complexity of integer programming problems. In *Optimization and Operations Research*, pages 161–172. Springer, 1978.

[37] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 184–193. IEEE, 1975.

[38] Steven Diamond, Reza Takapoui, and Stephen Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

[39] Reza Takapoui, Nicholas Moehle, Stephen Boyd, and Alberto Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, 93(1):2–12, 2020.

[40] Bingsheng He, Hai Yang, and Shengli Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and applications*, 106(2):337–356, 2000.

[41] Shengli Wang and Lizhi Liao. Decomposition method with a variable parameter for a class of monotone variational inequality problems. *Journal of optimization theory and applications*, 109(2):415–429, 2001.

[42] Brendt Wohlberg. Admm penalty parameter selection by residual balancing. *arXiv preprint arXiv:1704.06209*, 2017.

[43] Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[44] Bin Xiang, Jocelyne Elias, Fabio Martignon, and Elisabetta Di Nitto. Joint Planning of Network Slicing and Mobile Edge Computing: Models and Algorithms. *IEEE Transactions on Cloud Computing (under review)*, 2020.

[45] David Artuñedo Guillen, Bessem Sayadi, Pascal Bisson, Jean Phillippe Wary, Håkon Lonsethagen, Carles Antón, Antonio de la Oliva, Alexandros Kaloxylos, and Valerio Frascolla. 5GPPP Technology Board Working Group, 5G-IA's Trials Working Group: Edge Computing for 5G Networks. 2021.

[46] Francesco Spinelli and Vincenzo Mancuso. Toward enabled industrial verticals in 5g: A survey on MEC-based approaches to provisioning and flexibility. *IEEE Communications Surveys & Tutorials*, 23(1):596–630, 2020.

[47] Pasika Ranaweera, Madhusanka Liyanage, and Anca Delia Jurcut. Novel MEC based approaches for smart hospitals to combat COVID-19 pandemic. *IEEE Consumer Electronics Magazine*, 10(2):80–91, 2020.

[48] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2014.

Appendix A Problem Reformulation

Problem $\mathcal{P}0$ formulated in Section 3 cannot be solved directly and efficiently due to following reasons:

- We perform optimal routing (the routing path $\mathcal{R}^{kv}$ is a variable in our model, since many paths may exist from each request source node s^k to a generic node v in the network); furthermore, we must ensure that the properties of no-splitting, continuity and acyclicity are respected for our routing solution.
- Variables $\mathcal{R}^{kv}$ and q^{kv} are "intertwined": to find the optimal routing, the fraction of request processed at each node v should be known, and at the same time, to solve the optimal allocation for a request, the routing path should be known.
- $\mathcal{P}0$ contains indicator functions and constraints, e.g., (7), (10), (15), etc., which cannot be directly and easily processed by most solvers.

To deal with these challenging issues, we propose an equivalent reformulation of $\mathcal{P}0$, which can be solved very efficiently with the Branch and Bound method. Moreover, based on the reformulated problem, we propose an heuristic algorithm which can get near-optimal solutions in a short computing time.

A.1 Network Routing

To determine the routing path $\mathcal{R}^{kv}$, we first introduce a binary variable γ_e^{kv} defined as follows:

$$\gamma_e^{kv} = \begin{cases} 1, & \text{if } e \in \mathcal{R}^{kv}, \\ 0, & \text{otherwise,} \end{cases} \quad \forall k, \forall v, \forall e,$$

which indicates whether e is used in the routing path $\mathcal{R}^{kv}$ or not. Note that only if request k is processed on node v (i.e., $b^{kv} = 1$) and $v \neq s^k$, the corresponding routing path is defined. Then we have:

$$\begin{cases} \gamma_e^{ks^k} = 0, & \forall k, \forall e, \\ \gamma_e^{kv} \leqslant b^{kv}, & \forall k, \forall v, \forall e. \end{cases} \tag{57}$$

Based on the definitions introduced in the previous subsection, the traffic flow f_e^k can be transformed as:

$$f_e^k = \textstyle\sum_{v \in \mathcal{V}} \gamma_e^{kv} q^{kv}, \forall k, \forall e. \tag{58}$$

Now we need to simplify the traffic flow conservation constraint (see Eq. (6)). To this aim, and to simplify notation, we first introduce in the network topology a "dummy" entry node 0 which connects to all source nodes $s^k, k \in \mathcal{K}$. All requests are coming through this dummy node and going to each source node with volume λ^k, i.e. $f_e^k = 1, \forall k, \forall e \in \mathcal{F}$, where $\mathcal{F} = \{(0, s^k) \mid k \in \mathcal{K}\}$ is the dummy link set. Then, we extend the definition of Φ_v^- to $\Phi_v^- = \{(v', v) \in \mathcal{E} \cup \mathcal{F}\}$. Equation (6) is hence transformed as:

$$\sum_{e \in \Phi_v^-} f_e^k - \sum_{e \in \Phi_v^+} f_e^k = q^{kv}, \ \forall k, \forall v. \tag{59}$$

Correspondingly, we add the following constraints to the set $\mathcal{F}$ of dummy links:

$$\gamma_e^{kv} = \begin{cases} b^{kv}, & \text{if } e = (0, s^k) \\ 0, & \text{otherwise,} \end{cases} \quad \forall k, \forall v, \forall e \in \mathcal{F}. \tag{60}$$

The final stage of our procedure is the definition of the constraints that guarantee all desirable properties that a routing path must respect: the fact that a *single path* is used (a request piece is no more splittable), the flow conservation constraints that provide *continuity* to the chosen path, and finally the absence of *cycles* in the routing path $\mathcal{R}^{kv}$. We would like to highlight that the request k can be only split at source node s^k, and each portion of such traffic is destined to an edge node v, and this is the reason why we have multiple routing paths $\mathcal{R}^{kv}, v \in \{1, 2, \cdots\}$.

To this aim, we introduce the following conditions:

- For an arbitrary node v', the number of incoming links used by a path $\mathcal{R}^{kv}$ is one, and thus variables γ_e^{kv} should satisfy the following condition:

$$\textstyle\sum_{e \in \Phi_{v'}^-} \gamma_e^{kv} \leqslant 1, \ \forall k, \forall v, \forall v'. \tag{61}$$

- The flow conservation constraint (see Eq. (59)) implements the continuity of a traffic flow.
- Every routing path should have an end or a destination to avoid loops. This can be ensured by the following equation:

$$\gamma^{kv}_{(v,v')} = 0,\ \forall k, \forall (v, v') \in \mathcal{E}. \tag{62}$$

Satisfying them along with the constraints illustrated before can guarantee that such properties of the routing path are respected. The proof is as follows:

Proof. a) Substitute Eq. (58) into (59) and make the following transformation:

$$\sum_{v' \in \mathcal{V}} q^{kv'} \Big(\sum_{e \in \Phi^-_v} \gamma^{kv'}_e - \sum_{e \in \Phi^+_v} \gamma^{kv'}_e \Big) = q^{kv},\ \forall k, \forall v. \tag{63}$$

b) Based on constraints (57) and (60), we have:

$$\text{if } q^{kv'} = 0, \text{ then } \sum_{e \in \Phi^-_v} \gamma^{kv'}_e - \sum_{e \in \Phi^+_v} \gamma^{kv'}_e = 0.$$

c) From a) and b), we have:

$$\begin{cases} \sum_{e \in \Phi^-_v} \gamma^{kv}_e - \sum_{e \in \Phi^+_v} \gamma^{kv}_e = 1, \forall k, \forall v \mid q^{kv} > 0, \\ \sum_{e \in \Phi^-_v} \gamma^{kv'}_e - \sum_{e \in \Phi^+_v} \gamma^{kv'}_e = 0, \forall k, \forall v, \forall v' \neq v. \end{cases}$$

d) Based on c), constraint (60), conditions (61) and (62) can be written as:

$$\begin{cases} \sum_{e \in \Phi^-_{s^k}} \gamma^{kv}_e = 1,\ \forall k, \forall v \mid q^{kv} > 0, & (64) \\ \sum_{e \in \Phi^-_v} \gamma^{kv}_e = 1,\ \forall k, \forall v \mid q^{kv} > 0, & (65) \\ \sum_{e \in \Phi^-_v} \gamma^{kv'}_e = \sum_{e \in \Phi^+_v} \gamma^{kv'}_e \leqslant 1,\ \forall k, \forall v, \forall v' \neq v. & (66) \end{cases}$$

Their practical meaning is explained as follows:

- (64) ensures dummy link $(0, s^k)$ to be the zeroth link in any routing path $\mathcal{R}^{kv}$ if $q^{kv} > 0$,
- (65) ensures node v to be the end node of the last link in any routing path $\mathcal{R}^{kv}$ if $q^{kv} > 0$,
- (66) ensures that if $v \in \mathcal{E} \backslash \{v'\}$ is an intermediate node in a routing path $\mathcal{R}^{kv'}$, v should have only one incoming link and one outgoing link. It also indicates the continuity of a request flow.

e) Given a non-empty routing path $\mathcal{R}^{kv'}$ ($q^{kv'} > 0$), check its validity by using the above conditions:

- Let $v = s^k$ in (66), then based on (64), $\sum_{e \in \Phi^+_v} \gamma^{kv'}_e = 1$. Next, we assume $e_1 = (s^k, v_1)$ is the first link of the routing path $\mathcal{R}^{kv'}$, then $\gamma^{kv'}_{e_1} = 1$;
- If $v_1 = v'$, then the path is found, otherwise, we continue with the following steps:
- Let $v = v_1$ in (66), due to $\gamma^{kv'}_{e_1} = 1$, $\sum_{e \in \Phi^+_{v_1}} \gamma^{kv'}_e = 1$. Next, we assume $e_2 = (v_1, v_2)$ is the second link of $\mathcal{R}^{kv'}$, then $\gamma^{kv'}_{e_2} = 1$;
- We continue to check the path following the way as above the two steps until the final target $v_n = v'$ is reached, along the whole path $(s^k \rightarrow v') = (e_1, e_2, \cdots, e_n)$.

Thus, if all the conditions are satisfied, $\mathcal{R}^{kv'}$ must be a valid routing path having the three properties. □

Based on the above reformulation of routing, the flow conservation constraints can be further improved as follows:

$$\begin{cases} \sum_{e\in\Phi_v^-} \gamma_e^{kv} = b^{kv}, \qquad \forall k, \forall v, & (67)\\ \sum_{e\in\Phi_{v'}^-} \gamma_e^{kv} = \sum_{e\in\Phi_{v'}^+} \gamma_e^{kv}, \forall k, \forall v, \forall v' \neq v. & (68) \end{cases}$$

A.2 Link Latency

Based on the above definition of the routing variable γ_e^{kv}, we can rewrite constraint (8) as:

$$\begin{cases} q^{kv}\lambda^k - (1-\gamma_e^{kv})\Lambda^k < p_e^{kv}B_e, \\ p_e^{kv} \leqslant \gamma_e^{kv}, \end{cases} \forall k, \forall v, \forall e, \tag{69}$$

where $\Lambda^k = \lambda^k + c$ and $c = 1$ is a constant. Note that the term $(1-\gamma_e^{kv})$ permits to implement condition $e \in \mathcal{R}^{kv}$ in Eq. (8).

We now introduce variable h_e^{kv}, defined as follows:

$$h_e^{kv} = \frac{1}{p_e^{kv}B_e - q^{kv}\lambda^k + (1-\gamma_e^{kv})\Lambda^k}, \ \forall k, \forall v, \forall e. \tag{70}$$

This permits to transform Eq. (7) as $T_L^{kv} = \sum_{e\in\mathcal{E}} \gamma_e^{kv} h_e^{kv}$. We then need to linearize the product of the binary variable γ_e^{kv} and the continuous variable h_e^{kv}, and to this aim we introduce an auxiliary variable $g_e^{kv} = \gamma_e^{kv} h_e^{kv}$, thus also eliminate T_L^{kv}.

We first compute the value range of h_e^{kv} by considering the two cases: if $\gamma_e^{kv} = 0$, the range is $[(\Lambda^k)^{-1}, c^{-1}]$, where $c = \Lambda^k - \lambda^k$, and if $\gamma_e^{kv} = 1$, the range is $[B_e^{-1}, ((\beta^k - \alpha^k - d^k)\tau)^{-1}]$ based on constraint (4). In detail, if $\gamma_e^{kv} = 0$, then $p_e^{kv} = 0$ and the denominator of h_e^{kv} becomes $\Lambda^k - q^{kv}\lambda^k$, considering $q^{kv} \in [0,1]$, the range of h_e^{kv} is computed as $[(\Lambda^k)^{-1}, c^{-1}]$; if $\gamma_e^{kv} = 1$, the denominator of h_e^{kv} becomes $p^{kv}B_e - q^{kv}\lambda^k$, therefore, the upper limit of the denominator is B_e. Given that h_e^{kv} represents the single link latency which must be less than the allowed maximum latency $(\beta^k - \alpha^k - d^k)\tau$, therefore, the range of h_e^{kv} is $[B_e^{-1}, ((\beta^k - \alpha^k - d^k)\tau)^{-1}]$. Then, the linearization is performed by the following constraints.

$$\begin{cases} \gamma_e^{kv} B_e^{-1} \leqslant g_e^{kv} \leqslant \gamma_e^{kv}((\beta^k - \alpha^k - d^k)\tau)^{-1}, \\ (1-\gamma_e^{kv})(\Lambda^k)^{-1} \leqslant h_e^{kv} - g_e^{kv} \leqslant (1-\gamma_e^{kv})c^{-1}. \end{cases} \tag{71}$$

At the same time, the link latency is rewritten as:

$$T_L^{kv} = \sum\nolimits_{e\in\mathcal{E}} g_e^{kv}.$$

Since $p_e^{kvt} = \delta^{kvt} p_e^{kv}$ is the product of binary and continuous variables, we linearize it as:

$$\begin{cases} 0 \leqslant p_e^{kvt} \leqslant \delta^{kvt}, \\ 0 \leqslant p_e^{kv} - p_e^{kvt} \leqslant 1 - \delta^{kvt}, \end{cases} \forall k, \forall v, \forall t, \forall e. \tag{72}$$

Please remind that δ^{kvt} is a binary variable which is equal to 1 if $\xi_\star^k \leqslant t < \xi_\star^k + d^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil$, and 0 otherwise (see Eq. (10)). As we can see both upper and lower bounds of t are variables. We reformulate δ^{kvt} by the following constraints:

$$\begin{cases} \xi_\star^k - t \leqslant (\beta^k - d^k)(1 - \delta^{kvt}), \\ t - (\xi_\star^k + d^k + \pi_L^{kv}) < (\mathcal{T}_m - d^k + 1)(1 - \delta^{kvt}), \\ 0 \leqslant \sum_{t'\in\mathcal{T}} \delta^{kvt'} - (d^k + \pi_L^{kv}) \leqslant (\beta^k - \alpha^k)(1 - b^{kv}), \ \forall k, \forall v, \forall t, \\ 0 \leqslant \pi_L^{kv} - \frac{T_L^{kv}}{\tau} < 1, \end{cases} \tag{73}$$

where π_L^{kv} is an auxiliary integer variable for expanding the ceil operation over $\frac{T_L^{kv}}{\tau}$. The first and the second inequalities respectively enforce $\delta^{kvt} = 0$, when $t < \xi_\star^k$ and $t \geqslant (\xi_\star^k + d^k + \pi_L^{kv})$, which is the ending time of the link transmission. The third one enforces $\delta^{kvt} = 1$ when t is in the range $[\xi_\star^k, \xi_\star^k + d^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil)$ and $b^{kv} = 1$.

A.3 Processing Latency and Storage Provisioning

Equation (11) is a nonlinear indicator function of the variables r^{kv} and q^{kv}. To handle this issue, we first introduce an auxiliary variable b^{kv} to indicate whether request k is processed on node v. According to the definition of q^{kv}, we have the following constraint:

$$q^{kv} \leqslant b^{kv} \leqslant M q^{kv},\ \forall k, \forall v, \tag{74}$$

where $M > 0$ is a big value and such constraint implies that if $q^{kv} = 0$, the request k is not processed on node v, i.e. $b^{kv} = 0$. Based on the above, we can rewrite constraint (12) as:

$$\begin{cases} \eta^k q^{kv} \lambda^k - (1 - b^{kv}) < r^{kv} D_v, \\ r^{kv} \leqslant b^{kv}, \end{cases} \forall k, \forall v. \tag{75}$$

Note that the term $(1 - b^{kv})$ permits to implement condition $q^{kv} > 0$ in Eq. (12).

In equation (11), we observe that if $b^{kv} = 1$, we have:

$$\frac{1}{r^{kv} D_v - \eta^k q^{kv} \lambda^k} > \frac{1}{D_v} \geqslant \frac{1}{D_{max}},$$

where $D_{max} = \max_{v \in \mathcal{V}} D_v$, otherwise $r^{kv} D_v - \eta^k q^{kv} \lambda^k = 0$ resulting in $T_P^{kv} \to \infty$. To handle this case, we first define a new variable $T_{P'}^{kv}$ as follows:

$$T_{P'}^{kv} = \frac{1}{r^{kv} D_v - \eta^k q^{kv} \lambda^k + (1 - b^{kv}) D_{max}}. \tag{76}$$

From this equation, we have $b^{kv} = 1 \Rightarrow T_{P'}^{kv} = T_P^{kv} > \frac{1}{D_{max}}$ and $b^{kv} = 0 \Rightarrow T_{P'}^{kv} = \frac{1}{D_{max}}$, $T_P^{kv} = 0$. More in detail, this indicates that if a request k is accepted and processed on a set of nodes $\mathcal{V}_{sub} \subseteq \mathcal{V}$ (i.e., $b^{kv} = 1, v \in \mathcal{V}_{sub}$), the processing latency is determined by $\max_{v \in \mathcal{V}_{sub}} T_P^{kv} > \frac{1}{D_{max}}$, therefore, $T_{P'}^{kv}$ satisfies the related constraints and represents the exact processing latency when request k is accepted. Instead if k is rejected, then we have $z^{kt} = 0, b^{kv} = 0, \forall v, t$. Based on constraint (4) specifying that the ending time depends on the maximum latency and considering that a rational and meaningful request should satisfy $d^k + \left\lceil \frac{1}{D_{max} \tau} \right\rceil < \beta^k - \alpha^k$, we have $\xi_o^{kv} = d^k + \left\lceil \frac{T_{P'}^{kv}}{\tau} \right\rceil < \beta^k$. Therefore, $T_{P'}^{kv}$ is a valid representation for the processing latency and the reformulation has no influence on the solution of the optimization problem.

Since $r^{kvt} = \rho^{kvt} r^{kv}$ is the product of binary and continuous variables, we linearize it as:

$$\begin{cases} 0 \leqslant r^{kvt} \leqslant \rho^{kvt}, \\ 0 \leqslant r^{kv} - r^{kvt} \leqslant 1 - \rho^{kvt}, \end{cases} \forall k, \forall v, \forall t. \tag{77}$$

Recall that ρ^{kvt} is a binary variable which is equal to 1 if $\xi_\star^k + \left\lceil \frac{T_L^{kv}}{\tau} \right\rceil \leqslant t < \xi_o^{kv}$, and 0, otherwise (see Eq. (15)). We can see in ρ^{kvt} that both upper and lower bounds of t are variables. We reformulate ρ^{kvt} by the following constraints: The derivation is very similar to the one for δ^{kvt} (see inequality (73)) due to the similar definitions of the variables.

$$\begin{cases} \xi_\star^k + \pi_L^{kv} - t \leqslant (\beta^k - d^k)(1 - \rho^{kvt}), \\ t - \xi_o^{kv} < (\mathcal{T}_m - d^k + 1)(1 - \rho^{kvt}), \\ 0 \leqslant \sum_{t' \in \mathcal{T}} \rho^{kvt'} - (d^k + \pi_{P'}^{kv}) \leqslant (\beta^k - \alpha^k)(1 - b^{kv}), \\ 0 \leqslant \pi_{P'}^{kv} - \frac{T_{P'}^{kv}}{\tau} < 1, \end{cases} \forall k, \forall v, \forall t, \tag{78}$$

where $\pi_{P'}^{kv}$ is an auxiliary integer variable for expanding the ceil operation over $\frac{T_{P'}^{kv}}{\tau}$. Based on above, the deadline constraint (4) can be rewritten as:

$$\xi_{\star}^{k} + d^{k} + \pi_{L}^{kv} + \pi_{P'}^{kv} \leqslant \beta^{k}, \ \forall k, \forall v. \tag{79}$$

A.4 Final Reformulated Problem

Based on the above derivation, the reformulated optimization ($\mathcal{P}.\mathcal{R}1$) is written as follows:

$$\begin{aligned} \max \ & \sum_{t\in\mathcal{T}} \sum_{k\in\mathcal{K}} \left\{ \mu^{k} z^{kt} - \sum_{v\in\mathcal{V}} \left\{ r^{kvt} D_{v}\theta_{v} + \rho^{kvt} m^{k} \phi_{v} + \sum_{e\in\mathcal{E}} p_{e}^{kvt} B_{e}\psi_{e} \right\} \right\}, \\ \text{s.t.} \ & (1) \sim (3), (9), (13) \sim (14), (16), \\ & (57), (60) \sim (62), (67) \sim (79), \end{aligned} \tag{$\mathcal{P}.\mathcal{R}1$}$$

which is equivalent to the optimization ($\mathcal{P}0$). Since constraints (70) and (76) are quadratic while the others are linear, ($\mathcal{P}.\mathcal{R}1$) is a mixed-integer quadratically constrained programming (MIQCP) problem, for which commercial and open source solvers can be used, as we discussed in the numerical evaluation section.

Appendix B ADMM Solution Derivation

This appendix derives the analytic solution for $\boldsymbol{y}$-*update* in ADMM, i.e., the optimization in equation (39), which is recalled for the sake of clarity hereafter:

$$\boldsymbol{y}_{i+1} := \operatorname*{argmin}_{\boldsymbol{y}} \left\{ \mathcal{I}_{\mathcal{A}}(\boldsymbol{y}) + \frac{\sigma}{2} \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}} ||\boldsymbol{y}^{kv} - \boldsymbol{x}_{i+1}^{kv} - \boldsymbol{u}_{i}^{kv}||^{2} \right\}.$$

In order to solve the above $\boldsymbol{y}$-*update*, we split the l_2 norm $||\boldsymbol{y}^{kv} - \boldsymbol{x}_{i+1}^{kv} - \boldsymbol{u}_{i}^{kv}||^{2}$ and the indicator function $\mathcal{I}_{\mathcal{A}}(\boldsymbol{y})$, and separately update the independent copying variables ${}_{z}\boldsymbol{y}^{kv}$, ${}_{q}\boldsymbol{y}^{kv}$, ${}_{r}\boldsymbol{y}^{kv}$, ${}_{\rho}\boldsymbol{y}^{kv}$ and ${}_{p}\boldsymbol{y}^{kv}$. In the following, we present the detailed derivation of solution for each subproblem related to $\boldsymbol{y}^{kv}$.

• For ${}_{z}\boldsymbol{y}^{kv}$, which is a *copy* of variable $\mathring{\boldsymbol{z}}^{kv}$ related to the admission control of requests, we can rewrite the update as:

$$\begin{aligned} \min_{\{{}_{z}\boldsymbol{y}^{kv}\}} \ & \sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{V}} ||{}_{z}\boldsymbol{y}^{kv} - \mathring{\boldsymbol{z}}_{i+1}^{kv} - {}_{z}\boldsymbol{u}_{i}^{kv}||^{2}, \\ \text{s.t.} \ & {}_{z}\boldsymbol{y}^{kv} = \boldsymbol{z}^{k}, \ \forall k, \forall v. \end{aligned}$$

Based on the consensus constraint, the above minimization problem can be splitted into $|\mathcal{K}|$ independent unconstrained problems and each subproblem (k) can be written as:

$$\min_{\boldsymbol{z}^{k}} \ \sum_{v\in\mathcal{V}} ||\boldsymbol{z}^{k} - \mathring{\boldsymbol{z}}_{i+1}^{kv} - {}_{z}\boldsymbol{u}_{i}^{kv}||^{2}.$$

Taking the derivative of the above objective function w.r.t. $\boldsymbol{z}^{k}$ equal to 0, we could get the solution of $\boldsymbol{z}^{k}$. Since ${}_{z}\boldsymbol{y}^{kv} = \boldsymbol{z}^{k}, \forall k, \forall v$, the solution of ${}_{z}\boldsymbol{y}^{kv}$ is written as:

$$\begin{aligned} {}_{z}\boldsymbol{y}^{kv} &= \frac{1}{|\mathcal{V}|} \sum_{v'\in\mathcal{V}} (\mathring{\boldsymbol{z}}_{i+1}^{kv'} + {}_{z}\boldsymbol{u}_{i}^{kv'}) \\ &= \overline{\boldsymbol{z}}_{i+1}^{k} + {}_{z}\overline{\boldsymbol{u}}_{i}^{k}, \end{aligned} \tag{80}$$

where $\overline{\boldsymbol{z}}_{i+1}^{k}$ and ${}_{z}\overline{\boldsymbol{u}}_{i}^{k}$ denote respectively the two average terms.

• For ${}_{q}\boldsymbol{y}^{kv}$, which is a *copy* of variable $\mathring{q}^{kv}$ indicating the fractions of requests processed on different edge nodes, the update in (39) can be also splitted to $|\mathcal{K}|$ independent problems, and each subproblem (k) can be written as:

$$\min_{\{{}_{q}\boldsymbol{y}^{kv}\}} \ \sum_{v\in\mathcal{V}} ||{}_{q}\boldsymbol{y}^{kv} - \mathring{q}_{i+1}^{kv} - {}_{q}\boldsymbol{u}_{i}^{kv}||^{2},$$

$$\text{s.t.} \quad \sum\nolimits_{v\in\mathcal{V}} {}_q\boldsymbol{y}^{kv} = 0.$$

This subproblem is a convex optimization problem. We can use the method of Lagrange multipliers to optimally solve it. To do so, we first write the Lagrangian function as:

$$\mathcal{L}({}_q\boldsymbol{y}^{kv}, \lambda) = \sum_{v\in\mathcal{V}} ||{}_q\boldsymbol{y}^{kv} - \mathring{q}^{kv}_{i+1} - {}_q\boldsymbol{u}^{kv}_i||^2 + \lambda \sum_{v\in\mathcal{V}} {}_q\boldsymbol{y}^{kv},$$

where $\lambda \neq 0$ is the Lagrange multiplier. Then, the KKT conditions can be written as:

$$\begin{cases} \frac{\partial}{\partial_q \boldsymbol{y}^{kv}}\mathcal{L}({}_q\boldsymbol{y}^{kv}, \lambda) = 0, \\ \frac{\partial}{\partial\lambda}\mathcal{L}({}_q\boldsymbol{y}^{kv}, \lambda) = 0. \end{cases} \tag{81}$$

Based on the above equations, we can obtain the following solution:

$$\begin{aligned} {}_q\boldsymbol{y}^{kv} &= \mathring{q}^{kv}_{i+1} + {}_q\boldsymbol{u}^{kv}_i - \frac{1}{|\mathcal{V}|}\sum_{v'\in\mathcal{V}} \left\{ \mathring{q}^{kv'}_{i+1} + {}_q\boldsymbol{u}^{kv'}_i \right\} \\ &= \mathring{q}^{kv}_{i+1} + {}_q\boldsymbol{u}^{kv}_i - \overline{q}^{k}_{i+1} - {}_q\overline{\boldsymbol{u}}^{k}_i, \end{aligned} \tag{82}$$

where $\overline{q}^{k}_{i+1}$ and ${}_q\overline{\boldsymbol{u}}^{k}_i$ denote respectively the two average terms.

• For ${}_r\boldsymbol{y}^{kv}$, which is a *copy* of variable $\boldsymbol{r}^{kv}$ representing the computation capacities of edge nodes allocated to different requests, in a similar way, we could split the update in (39) to $|\mathcal{V}|$ independent unconstrained problems, and each subproblem (v) is written as:

$$\min_{\{{}_r\boldsymbol{y}^{kv}\}} \quad \mathcal{I}_{{}_r\mathcal{A}^v}(\{{}_r\boldsymbol{y}^{kv}\}) + \frac{\sigma}{2}\sum_{k\in\mathcal{K}} ||{}_r\boldsymbol{y}^{kv} - \boldsymbol{r}^{kv}_{i+1} - {}_r\boldsymbol{u}^{kv}_i||^2,$$

where ${}_r\mathcal{A}^v = \{{}_r\boldsymbol{y}^{kv} \mid \sum_{k\in\mathcal{K}} {}_r\boldsymbol{y}^{kv} \leqslant \mathbf{1}, {}_r\boldsymbol{y}^{kv} \geqslant \mathbf{0}\}$ corresponding to the constraint (31) (we recall its formulation: $\sum_{k\in\mathcal{K}} \boldsymbol{r}^{kv} \leqslant \mathbf{1}$) which is the reservation constraint of an edge node computation capacity. We can simplify the above minimization problem by introducing ${}_r\overline{\boldsymbol{y}}^{v} = \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}} {}_r\boldsymbol{y}^{kv}$. The derivation is detailed as follows.

The above minimization can be rewritten as:

$$\begin{aligned} \min_{{}_r\boldsymbol{y}^{kv}, {}_r\overline{\boldsymbol{y}}^{v}} \quad & \mathcal{I}_{{}_r\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_r\overline{\boldsymbol{y}}^{v}) + \frac{\sigma}{2}\sum_{k\in\mathcal{K}} ||{}_r\boldsymbol{y}^{kv} - \boldsymbol{r}^{kv}_{i+1} - {}_r\boldsymbol{u}^{kv}_i||^2, \\ \text{s.t.} \quad & {}_r\overline{\boldsymbol{y}}^{v} = \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}} {}_r\boldsymbol{y}^{kv}, \end{aligned}$$

where ${}_r\overline{\mathcal{A}}^v = \{\boldsymbol{x} \mid \mathbf{0} \leqslant \boldsymbol{x} \leqslant \mathbf{1}\}$ corresponding to set ${}_r\mathcal{A}^v$, which represents the constraint $\mathbf{0} \leqslant |\mathcal{K}|{}_r\overline{\boldsymbol{y}}^{v} = \sum_{k\in\mathcal{K}} {}_r\boldsymbol{y}^{kv} \leqslant \mathbf{1}$. Minimizing over ${}_r\boldsymbol{y}^{kv}$ with ${}_r\overline{\boldsymbol{y}}^{v}$ fixed and following the same way of solving ${}_q\boldsymbol{y}^{kv}$, we have the solution:

$$\begin{aligned} {}_r\boldsymbol{y}^{kv} &= \boldsymbol{r}^{kv}_{i+1} + {}_r\boldsymbol{u}^{kv}_i + {}_r\overline{\boldsymbol{y}}^{v} - \frac{1}{|\mathcal{K}|}\sum_{k'\in\mathcal{K}} \left\{ \boldsymbol{r}^{k'v}_{i+1} + {}_r\boldsymbol{u}^{k'v}_i \right\} \\ &= \boldsymbol{r}^{kv}_{i+1} + {}_r\boldsymbol{u}^{kv}_i + {}_r\overline{\boldsymbol{y}}^{v} - \overline{\boldsymbol{r}}^{v}_{i+1} - {}_r\overline{\boldsymbol{u}}^{v}_i, \end{aligned} \tag{83}$$

where $\overline{\boldsymbol{r}}^{v}_{i+1}$ and ${}_r\overline{\boldsymbol{u}}^{v}_i$ denote respectively the two average terms. Then, substituting (83) back into the minimization, we get the following unconstrained problem:

$$\min_{{}_r\overline{\boldsymbol{y}}^{v}} \quad \mathcal{I}_{{}_r\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_r\overline{\boldsymbol{y}}^{v}) + |\mathcal{K}|\frac{\sigma}{2}||{}_r\overline{\boldsymbol{y}}^{v} - \overline{\boldsymbol{r}}^{v}_{i+1} - {}_r\overline{\boldsymbol{u}}^{v}_i||^2. \tag{84}$$

Then, the update of ${}_r\boldsymbol{y}^{kv}$ is reduced to an optimization over the variable ${}_r\overline{\boldsymbol{y}}^{v}$.

• For ${}_\rho\boldsymbol{y}^{kv}$, which is a *copy* of variable $\mathring{\boldsymbol{\rho}}^{kv}$ related to the computation capacity and storage provisioning of edge nodes, the update has the similar structure as the one of ${}_r\boldsymbol{y}^{kv}$. Following the same procedure of ${}_r\boldsymbol{y}^{kv}$ update, the problem can be splitted to $|\mathcal{V}|$ independent unconstrained problems. We first introduce a variable ${}_\rho\overline{\boldsymbol{y}}^v = \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}} {}_\rho\boldsymbol{y}^{kv}$ and obtain the solution based on ${}_\rho\overline{\boldsymbol{y}}^v$:

$$\begin{aligned}{}_\rho\boldsymbol{y}^{kv} &= \mathring{\boldsymbol{\rho}}^{kv}_{i+1} + {}_\rho\boldsymbol{u}^{kv}_i + {}_\rho\overline{\boldsymbol{y}}^v - \frac{1}{|\mathcal{K}|}\sum_{k'\in\mathcal{K}}\left\{\mathring{\boldsymbol{\rho}}^{k'v}_{i+1} + {}_\rho\boldsymbol{u}^{k'v}_i\right\}\\ &= \mathring{\boldsymbol{\rho}}^{kv}_{i+1} + {}_\rho\boldsymbol{u}^{kv}_i + {}_\rho\overline{\boldsymbol{y}}^v - \overline{\boldsymbol{\rho}}^v_{i+1} - {}_\rho\overline{\boldsymbol{u}}^v_i,\end{aligned} \tag{85}$$

where $\overline{\boldsymbol{\rho}}^v_{i+1}$ and ${}_\rho\overline{\boldsymbol{u}}^v_i$ denote respectively the two average terms. ${}_\rho\overline{\boldsymbol{y}}^v$ can be computed through optimizing the following unconstrained problem:

$$\min_{{}_\rho\overline{\boldsymbol{y}}^v} \quad \mathcal{I}_{{}_\rho\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_\rho\overline{\boldsymbol{y}}^v) + |\mathcal{K}|\frac{\sigma}{2}||{}_\rho\overline{\boldsymbol{y}}^v - \overline{\boldsymbol{\rho}}^v_{i+1} - {}_\rho\overline{\boldsymbol{u}}^v_i||^2, \tag{86}$$

where ${}_\rho\overline{\mathcal{A}}^v = \{\boldsymbol{x}\,|\,\boldsymbol{0} \leqslant \boldsymbol{x} \leqslant \boldsymbol{S}_v\}$ corresponding to constraint (32) (here we recall its formulation, that is $\sum_{k\in\mathcal{K}}\mathring{\boldsymbol{\rho}}^{kv} \leqslant \boldsymbol{S}_v$) which is the reservation constraint of an edge node storage capacity.

• For ${}_p\boldsymbol{y}^{kv}$, which is a *copy* of variable $\boldsymbol{p}^{kv}$ representing the fractions of link bandwidth sliced to different requests, following the same procedure of ${}_r\boldsymbol{y}^{kv}$ update, we first introduce a variable ${}_p\overline{\boldsymbol{y}} = \frac{1}{|\mathcal{K}||\mathcal{V}|}\sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}} {}_p\boldsymbol{y}^{kv}$ and obtain the solution based on ${}_p\overline{\boldsymbol{y}}$:

$$\begin{aligned}{}_p\boldsymbol{y}^{kv} &= \boldsymbol{p}^{kv}_{i+1} + {}_p\boldsymbol{u}^{kv}_i + {}_p\overline{\boldsymbol{y}} - \frac{1}{|\mathcal{K}||\mathcal{V}|}\sum_{k'\in\mathcal{K}}\sum_{v'\in\mathcal{V}}\left\{\boldsymbol{p}^{k'v'}_{i+1} + {}_p\boldsymbol{u}^{k'v'}_i\right\}\\ &= \boldsymbol{p}^{kv}_{i+1} + {}_p\boldsymbol{u}^{kv}_i + {}_p\overline{\boldsymbol{y}} - \overline{\boldsymbol{p}}_{i+1} - {}_p\overline{\boldsymbol{u}}_i,\end{aligned} \tag{87}$$

where $\overline{\boldsymbol{p}}_{i+1}$ and ${}_p\overline{\boldsymbol{u}}_i$ denote respectively the two average terms. ${}_p\overline{\boldsymbol{y}}$ can be computed through optimizing the following unconstrained problem:

$$\min_{{}_p\overline{\boldsymbol{y}}} \quad \mathcal{I}_{{}_p\overline{\mathcal{A}}}(|\mathcal{K}||\mathcal{V}|{}_p\overline{\boldsymbol{y}}) + |\mathcal{K}||\mathcal{V}|\frac{\sigma}{2}||{}_p\overline{\boldsymbol{y}} - \overline{\boldsymbol{p}}_{i+1} - {}_p\overline{\boldsymbol{u}}_i||^2, \tag{88}$$

where ${}_p\overline{\mathcal{A}} = \{\boldsymbol{x}\,|\,\boldsymbol{0} \leqslant \boldsymbol{x} \leqslant \boldsymbol{1}\}$ corresponding to the constraint (33) (we report for clarity its formulation here: $\sum_{k\in\mathcal{K}}\sum_{v\in\mathcal{V}}\boldsymbol{p}^{kv} \leqslant \boldsymbol{1}$) which is the reservation constraint of a link capacity.

Based on the above simplified updates for $\boldsymbol{y}^{kv}$, substituting equations (80), (82), (83), (85) and (87) into (40) (whose expression is recalled here: $\boldsymbol{u}^{kv}_{i+1} := \boldsymbol{u}^{kv}_i + \boldsymbol{x}^{kv}_{i+1} - \boldsymbol{y}^{kv}_{i+1}$), we have:

$$\boldsymbol{u}^{kv}_{i+1} = \begin{bmatrix}\mathring{\boldsymbol{z}}^{kv}_{i+1} - {}_z\boldsymbol{y}^{kv}_{i+1}\\ \overline{q}^{k}_{i+1}\\ \overline{\boldsymbol{r}}^v_{i+1} - {}_r\overline{\boldsymbol{y}}^v_{i+1}\\ \overline{\boldsymbol{\rho}}^v_{i+1} - {}_\rho\overline{\boldsymbol{y}}^v_{i+1}\\ \overline{\boldsymbol{p}}_{i+1} - {}_p\overline{\boldsymbol{y}}_{i+1}\end{bmatrix} + \begin{bmatrix}{}_z\boldsymbol{u}^{kv}_i\\ {}_q\overline{\boldsymbol{u}}^k_i\\ {}_r\overline{\boldsymbol{u}}^v_i\\ {}_\rho\overline{\boldsymbol{u}}^v_i\\ {}_p\overline{\boldsymbol{u}}_i\end{bmatrix} = \begin{bmatrix}{}_z\boldsymbol{u}^{kv}_{i+1}\\ {}_q\overline{\boldsymbol{u}}^k_{i+1}\\ {}_r\overline{\boldsymbol{u}}^v_{i+1}\\ {}_\rho\overline{\boldsymbol{u}}^v_{i+1}\\ {}_p\overline{\boldsymbol{u}}_{i+1}\end{bmatrix}. \tag{89}$$

Equation (89) shows that the values of components (q, r, ρ, p) in $\boldsymbol{u}^{kv}_{i+1}$ are equal to their corresponding average values, respectively. Thus, we further simplify equations (82), (83), (85) and (87) as follows:

$$\begin{cases} {}_q\boldsymbol{y}^{kv}_{i+1} = \mathring{q}^{kv}_{i+1} - \overline{q}^k_{i+1}, & (90)\\ {}_r\boldsymbol{y}^{kv}_{i+1} = \boldsymbol{r}^{kv}_{i+1} - \overline{\boldsymbol{r}}^v_{i+1} + {}_r\overline{\boldsymbol{y}}^v, & (91)\\ {}_\rho\boldsymbol{y}^{kv}_{i+1} = \mathring{\boldsymbol{\rho}}^{kv}_{i+1} - \overline{\boldsymbol{\rho}}^v_{i+1} + {}_\rho\overline{\boldsymbol{y}}^v, & (92)\\ {}_p\boldsymbol{y}^{kv}_{i+1} = \boldsymbol{p}^{kv}_{i+1} - \overline{\boldsymbol{p}}_{i+1} + {}_p\overline{\boldsymbol{y}}, & (93)\end{cases}$$

where the variables ${}_r\overline{\boldsymbol{y}}^v, {}_\rho\overline{\boldsymbol{y}}^v, {}_p\overline{\boldsymbol{y}}$ are determined by the following optimizations based on above derivations:

$$\begin{cases} {}_r\overline{\boldsymbol{y}}^v_{i+1} = \underset{{}_r\overline{\boldsymbol{y}}^v}{\operatorname{argmin}} \left\{ \mathcal{I}_{{}_r\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_r\overline{\boldsymbol{y}}^v) + |\mathcal{K}|\frac{\sigma}{2}||{}_r\overline{\boldsymbol{y}}^v - \overline{\boldsymbol{r}}^v_{i+1} - {}_r\overline{\boldsymbol{u}}^v_i||^2 \right\}, & (94) \\ {}_\rho\overline{\boldsymbol{y}}^v_{i+1} = \underset{{}_\rho\overline{\boldsymbol{y}}^v}{\operatorname{argmin}} \left\{ \mathcal{I}_{{}_\rho\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_\rho\overline{\boldsymbol{y}}^v) + |\mathcal{K}|\frac{\sigma}{2}||{}_\rho\overline{\boldsymbol{y}}^v - \overline{\boldsymbol{\rho}}^v_{i+1} - {}_\rho\overline{\boldsymbol{u}}^v_i||^2 \right\}, & (95) \\ {}_p\overline{\boldsymbol{y}}_{i+1} = \underset{{}_p\overline{\boldsymbol{y}}}{\operatorname{argmin}} \left\{ \mathcal{I}_{{}_p\overline{\mathcal{A}}}(|\mathcal{K}||\mathcal{V}|{}_p\overline{\boldsymbol{y}}) + |\mathcal{K}||\mathcal{V}|\frac{\sigma}{2}||{}_p\overline{\boldsymbol{y}} - \overline{\boldsymbol{p}}_{i+1} - {}_p\overline{\boldsymbol{u}}_i||^2 \right\}. & (96) \end{cases}$$

In the above optimizations (94), (95) and (96), all of them are composed of two components: i) the indicator function on decision variable which can be regarded as the constraint, e.g., $\mathcal{I}_{{}_r\overline{\mathcal{A}}^v}(|\mathcal{K}|{}_r\overline{\boldsymbol{y}}^v)$, ii) the ℓ_2 norm which represents the Euclidean distance, e.g., $||{}_r\overline{\boldsymbol{y}}^v - (\overline{\boldsymbol{r}}^v_{i+1} + {}_r\overline{\boldsymbol{u}}^v_i)||^2$. Thus, they are equivalent to three different Euclidean projections onto the convex sets ${}_r\overline{\mathcal{A}}^v$, ${}_\rho\overline{\mathcal{A}}^v$ and ${}_p\overline{\mathcal{A}}$, which have the following closed form solutions:

$$\begin{cases} {}_r\boldsymbol{y}^{kv}_{i+1} = \boldsymbol{r}^{kv}_{i+1} - \overline{\boldsymbol{r}}^v_{i+1} + \mathbf{P}_{[\mathbf{0},\mathbf{1}/|\mathcal{K}|]}(\overline{\boldsymbol{r}}^v_{i+1} + {}_r\overline{\boldsymbol{u}}^v_i) & (97) \\ {}_\rho\boldsymbol{y}^{kv}_{i+1} = \mathring{\boldsymbol{\rho}}^{kv}_{i+1} - \overline{\boldsymbol{\rho}}^v_{i+1} + \mathbf{P}_{[\mathbf{0},S_v/|\mathcal{K}|]}(\overline{\boldsymbol{\rho}}^v_{i+1} + {}_\rho\overline{\boldsymbol{u}}^v_i) & (98) \\ {}_p\boldsymbol{y}^{kv}_{i+1} = \boldsymbol{p}^{kv}_{i+1} - \overline{\boldsymbol{p}}_{i+1} + \mathbf{P}_{[\mathbf{0},\mathbf{1}/(|\mathcal{K}||\mathcal{V}|)]}(\overline{\boldsymbol{p}}_{i+1} + {}_p\overline{\boldsymbol{u}}_i) & (99) \end{cases}$$

where $\mathbf{P}_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{e}) = (\min\{\max\{\boldsymbol{a}_j, \boldsymbol{e}_j\}, \boldsymbol{b}_j\})_{j=1}^{\dim(\boldsymbol{e})}$ denotes the projection of $\boldsymbol{e}$ on box $[\boldsymbol{a}, \boldsymbol{b}]$, and $\overline{\boldsymbol{z}}^k_{i+1}, \overline{q}^k_{i+1}, \overline{\boldsymbol{r}}^v_{i+1}, \overline{\boldsymbol{\rho}}^v_{i+1}, \overline{\boldsymbol{p}}_{i+1}$ are the average value for each item of $\boldsymbol{x}^{kv}_{i+1}$. Note that a similar representation is applicable to $\boldsymbol{u}^{kv}_{i+1}$.