Heuristic Distribution of Latency-Sensitive Tasks in Multi-Access Edge Computing Systems

Guilherme Iecker Ricardo, Amal Benhamiche, Nancy Perrot, and Yannick Carlinet

Orange Labs, Châtillon, France
Email: {guilhermeiecker.ricardo, amal.benhamiche, nancy.perrot, yannick.carlinet}@orange.com

***Abstract*—Multi-Access Edge Computing (MEC) paradigm has been widely studied as a potential solution to cope with the challenges emerging from new generations of mobile networks. By processing applications' data closer to the users, service providers are able to offload origin servers and their underlying network infrastructure, which consequently reduces users' experienced latency. In this paper, we consider internet-based applications with strict latency tolerance which are primarily enabled by the MEC architecture. Moreover, nodes at the edge may host application-related tasks as well as assist in their provision. We introduce the Task Distribution Problem (TDP), where the objective is to maximize the overall Quality of Service (QoS) based on the achieved throughput while ensuring that tasks' latency requirements are satisfied. The TDP is modeled as an Integer Programming problem, taking into account three components: (i) tasks' priority assignment, (ii) placement and (iii) routing through the MEC network. We propose to approach the problem through two different heuristics: a greedy replacement algorithm and a streaming algorithm. In our experiments, we evaluate the algorithms' performance by showing numerical results across different experimental settings. We observe that, for the tested scenarios, our techniques provide a good trade-off between run time and high performance.**

***Index Terms*—Edge Computing, Resource Allocation, Quality of Service, Operations Research, Combinatorial Optimization, Greedy Algorithm, Streaming Algorithm**

I. Introduction

The unprecedented increasing trend in terms of mobile connectivity and the evolution of data consumption profiles have imposed a series of challenges to the next generation of mobile networks [1]. For example, they should be able to accommodate an immense and heterogeneous number of connected devices, ranging from personal smartphones to Internet of Things (IoT) equipment (smart appliances, sensors, etc.) [2]. Not only must the infrastructure be able to handle intense traffic and data processing, it must also guarantee that specific requirements, e.g., in terms of latency and bandwidth, are met in order to enable disruptive technologies and innovative applications. It has been widely discussed in the literature that shifting from the classic Cloud Computing paradigm to the Multi-Access (Mobile) Edge Computing (MEC) paradigm may significantly contribute to address such challenges [3].

The authors are with the Department of Mathematical Models for Optimization and Performance Evalualtion, Orange Innovation, Châtillon, 92012 France. This work is supported by the European Union H2020 Project DEDICAT 6G under grant number 101016499. Corresponding author's contact information: http://www.guilhermeir.github.io

If we consider the MEC reference architecture [4], roughly speaking, the idea is to move data processing and storage from remote cloud servers to network nodes that are located at the "edge", closer to the application's users. The distribution of computation across the network offers the possibility to offload both origin servers and the underlying network infrastructure. In the context of Industry 4.0, Smart Warehouse (SW) systems are characterized by the combination of all the aforementioned challenges in an effort to promote digitization and automation of industrial processes. They have been considered an important use case in most envisioned next generation architecture designs, e.g., the EU-funded DEDICAT 6G project [5]. In particular, SW systems must provide real-time human-machine interaction services with strict latency requirements, e.g., automatically guided vehicles, timely computer-aided industrial operations (e.g., assisted with virtual and augmented reality technologies), etc. A MEC-powered network design is a key enabler of such a system through back-end computation offload and opportunistic networking.

A. Related Work

MEC architectures and applications are comprehensively summarized in [6], [7]. Resource allocation on MEC systems has been addressed in numerous variants, e.g., joint minimization of task completion time and energy cost [8], cross-layer multiple resource allocation [9], auction-based blockchain applications [10], and QoS for vehicular systems [11]. Some industrial solutions include, for example, efficient routing for energy-balanced consumption in heterogeneous Industrial IoT (IIoT) [12] and energy-efficient scheduling guaranteeing delay requirements [13]. Authors in [14] propose a full model for joint optimization of service placement and routing capabilities targeting low latency applications. Moreover, in [15], the authors propose an optimization model and greedy algorithm to perform resource allocation in MEC network for the edge-gaming application, considering energy saving and high intensity computation. The model considered in [16] for the cost minimization problem in OFDM two-layer networks share many similarities to the modeling proposed in this paper. The streaming optimization framework [17] is the foundation of one of our proposed techniques. A seminal work in this area was proposed in [18], which discusses streaming algorithms for the optimization of submodular functions under cardinality constraints.

B. Main Contributions

We outline our main contributions and present the paper organization as follows:

- We propose a model for a MEC network operating under latency-sensitive requirements in Section II. We model tasks' latency as a linear function of the network's links rate in order to characterize such requirements.
- We introduce the Task Distribution Problem (TDP) in Section III as a throughput maximization problem. We provide theoretical results on its complexity and discuss practical aspects of its exact solution.
- In Section IV, we propose two replacement algorithms to address the problem. The first one is based on greedy decisions and, even though it is more computationally demanding, it has optimality guarantees under uniform scenarios. The second one has a shorter run time and it demands a single pass over the problem's input.
- We discuss experimental results in Section V, showing the algorithms' relative performance in practice.

II. System Model and Assumptions

Consider a mobile network consisting of a set $\mathcal{U}$ of user equipment (UEs), where each UE is associated with a single mobile Base Station (BS) from a set $\mathcal{B}$. Each BS is equipped with a MEC host for application-oriented data storage, processing, and routing. For simplicity, we assume that BSs are able to perfectly handle all transmissions from and to their associated UEs, such that UE-BS communication will be transparent to our model. In the transport network, BSs' traffic often converges at sink nodes/gateways also equipped with MEC hosts, which we refer to as "near-edge" nodes (NENs). We represent the set of NENs by $\mathcal{N}$. MEC hosts may be interconnected (e.g. via Mp3 interface), forming the *MEC network*. The logical topology of the MEC network is represented by a graph $G = (\mathcal{H}, \mathcal{E})$, where nodes $\mathcal{H} = \mathcal{B} \cup \mathcal{N}$ are the MEC hosts (BSs and NENs) and there is an edge in $\mathcal{E} \subseteq \mathcal{H} \times \mathcal{H}$ for each connected pair of MEC hosts.

We refer to applications and services indistinguishably as *tasks*. As in some related work (e.g., [19]), we consider a fixed catalog $\mathcal{T}$ of tasks that are pre-installed at every MEC host. Each UE may place requests for tasks to its associated BS, which will, in turn, forward the request to the *Multi-Access Edge Orchestrator* (MEO), a centralized control intelligence aware of the entire network's infrastructure and available resources. The system has a periodic operation split into *setup* phase and *main* phase. In the setup phase, the MEO, upon receiving the set of requests $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{T}$, determines how resources should be allocated for their provision. It also assigns one MEC host to be the *provider* of each request and the route through which tasks' data will be forwarded. Then, in the main phase, tasks' data is effectively exchanged between UEs and providers and eventual new requests are placed to be handled on the setup phase of the next period.

A. System Operation

In the setup phase, the MEO must determine a *provision plan*, which consists of three components.

1) Priority assignment: Consider a set $\mathcal{P}$ of priorities. One priority must be assigned to each request, which is associated to a specific transmission performance level (and, consequently, to the demanded amount of resources). We measure performance in terms of the achieved average throughput, which is responsible for providing UEs with a smoother, uninterrupted experience [20]. When priority p is assigned to request r, the associated task's data transmission will achieve an average throughput of T_p^r.

2) Request allocation: Each request r must be allocated to a MEC host (its provider). The provision of each request consumes a portion of the host's computational resources, e.g. in terms of RAM or CPU cycles, and it depends of the request's assigned priority. Each MEC host may be the provider of multiple requests (at different priorities) as long as it has enough available resources. When a request r is assigned to a MEC host h at a priority p, it consumes D_p^r of host h's total available computational resources C_h.

3) Request routing: If the MEO allocates a request to a NEN, then it must also provide a route on the MEC network connecting the request's UE's BS and the actual NEN provider. We represent a request route simply as a path $\{(s, h), \ldots, (h', d)\}$ on G, i.e., a cycle-free sequence of distinct edges connecting a source node s to a destination node d, which we henceforth refer to as a *flow*. We define the set of (equivalent) $s - d$ flows, denoted by $\mathcal{F}_{s,d}$, as a set of all flows sharing source and destination nodes. For the MEC network, we define the set of all considered flows as $\mathcal{F} = \{\mathcal{F}_{s,d} : \forall s \in \mathcal{B}, \forall d \in \mathcal{H}\}$.

In order to determine the provision plan, we capture each of its three components with the decision variable

$$\boldsymbol{\lambda} \in \{0,1\}^{\mathcal{R} \times \mathcal{F} \times \mathcal{P}}, \tag{1}$$

indicating whether request $r \in \mathcal{R}$ is assigned to flow $f \in \mathcal{F}$ at priority $p \in \mathcal{P}$ (i.e., $\lambda_{f,p}^r = 1$) or not (i.e., $\lambda_{f,p}^r = 0$).

B. Latency Model

MEC-network routing naturally produces a *latency* overhead in the tasks' data transmissions due the potential multiple hops. This latency may be an obstacle for some sensitive tasks and must be avoided in the provision plan. In this work, we consider a latency model based on the end-to-end delay definition [21], i.e., the time to transmit application packets from the request's UE's BS to its provider. We assume that a request's latency consists uniquely of its queuing delay component[1], which is fundamentally impacted by the total throughput at each routing link.

[1]As introduced in [21], the end-to-end delay has four components: transmission, propagation, processing, and queuing delays. We justify this assumption by considering a roughly homogeneous network (e.g., packets with the same size, links with equal capacity, etc.), so, in order to simplify our analysis, we can disregard sources of delay other than the queuing delay.

Consider that the total rate over a link $e \in \mathcal{E}$ is simply given by the sum of the average throughput over all requests routed through it, i.e.,

$$\text{RATE}_e(\boldsymbol{\lambda}) \triangleq \sum_{r\in\mathcal{R}} \sum_{f\in\mathcal{F}} \sum_{p\in\mathcal{P}} \mathbb{1}_{e\in f} T^r_p \lambda^r_{f,p}, \tag{2}$$

where $\mathbb{1}_x$ indicates the occurrence of event x. As in some related work (e.g., [22]), we approximate the latency (i.e., the queuing delay) of a request r by a linear function of the rate [2] on its routing links, i.e.,

$$\text{LATENCY}^r(\boldsymbol{\lambda}) \triangleq \sum_{f\in\mathcal{F}} \sum_{p\in\mathcal{P}} \lambda^r_{f,p} \sum_{e\in f} \alpha_e \text{RATE}_e(\boldsymbol{\lambda}) + \beta_e \tag{3}$$

where $\text{RATE}_e(\cdot)$ is given by (2) and $\alpha_e, \beta_e \in \mathbb{R}_+$ are the line coefficients for edge e. In summary, latency is impacted by two factors: (i) The number of links forwarding the request's data and (ii) the rate over all requests traversing each link.

III. The Task Distribution Problem

Considering the system model presented in Section II, the Task Distribution Problem (TDP) aims at determining a provision plan that (i) maximizes the system's performance and (ii) meets the requests' latency tolerance. In Problem 1, we present the TDP as an Integer Programming (IP) problem.

Problem 1 (Task Distribution Problem – TDP)**.**

$$\underset{\boldsymbol{\lambda}}{\text{maximize}} \quad \text{QOS}(\boldsymbol{\lambda}) \triangleq \frac{1}{|\mathcal{R}|} \sum_{r\in\mathcal{R}} \sum_{f\in\mathcal{F}} \sum_{p\in\mathcal{P}} T^r_p \lambda^r_{f,p} \tag{4}$$

subject to

$$\sum_{f\in\mathcal{F}} \sum_{p\in\mathcal{P}} \mathbb{1}_{s(f)=b(r)} \lambda^r_{f,p} = 1, \qquad \forall r \in \mathcal{R} \tag{5}$$

$$\sum_{r\in\mathcal{R}} \sum_{f\in\mathcal{F}^r} \sum_{p\in\mathcal{P}} \mathbb{1}_{d(f)=h} D^r_p \lambda^r_{f,p} \le C_h, \quad \forall h \in \mathcal{H} \tag{6}$$

$$\text{LATENCY}^r(\boldsymbol{\lambda}) \le L^r, \qquad \forall r \in \mathcal{R} \tag{7}$$

$$\boldsymbol{\lambda} \in \{0,1\}^{\mathcal{R}\times\mathcal{F}\times\mathcal{P}} \tag{8}$$

The elements of Problem 1 are described as follows:

- Objective function: The QoS function (4) is defined in terms of the average throughput over all requests.
- Singularity constraints: In (5), to every request $r \in \mathcal{R}$, we assign exactly one flow and one priority. Notice that, by multiplying by $\mathbb{1}_{s(f)=b(r)}$, we are enforcing that the chosen flow f has its source node $s(f) \in \mathcal{B}$ coinciding with the BS $b(r) \in \mathcal{B}$ associated with request r's UE.
- Computational capacity constraints: Constraints (6) ensure that the computational limitation of each host is met. We multiply by $\mathbb{1}_{d(f)=h}$ in order to guarantee that we are considering only if flow f's destination host $d(f)$ consumes resources of host h.
- Latency requirement constraints: In constraints (7), we ensure that request r's latency, as defined in (3), meets its threshold L^r.

[2]The system is stable if the total throughput over every link is bounded by the link's capacity [21]. We assume that all links have enough capacity to accommodate the maximum priority throughput.

Proposition III.1. Problem 1 is $\mathcal{NP}$-Hard.

Due to space limitations, we present the full proof of Proposition III.1 in Appendix A. The idea is to reduce the Multiple Knapsack Problem, which is $\mathcal{NP}$-Hard [23], to a particular instance of Problem 1, where requests enjoy sufficiently high latency tolerance such that routing can be disregarded in the problem's solution. A corollary of such a reduction is that TDP's decision version is $\mathcal{NP}$-Complete in the strong sense, so it does not admit a polynomial time approximation scheme (PTAS).

Remark III.1. Because latency constrains (7) can be "linearized" through a succession of substitutions, we can find its exact solution through traditional integer programming methods. Moreover, we can obtain an upper bound to the linear version of Problem 1 if we solve its continuous relaxation, i.e., relaxing the integrality of variables $\boldsymbol{\lambda}$. Then, Problem 1's relaxation can be translated to a Linear Programming (LP) problem, which can be efficiently solved.

The setup time for the MEO to determine the provision plan should be as short as possible, which makes most IP exact solution methods unfeasible to be used in practice. In the next section, we propose to tackle the TDP with approximate algorithms and discuss some theoretical optimality results.

IV. Approximate Algorithms

In this section, we introduce two heuristic methods to approach Problem 1: (1) the Greedy Replacement Algorithm and (2) the Streaming Replacement Algorithm.

A. Set Formulation

We build a ground set $V = \mathcal{R} \times \mathcal{F} \times \mathcal{P}$, which each element $v = (r, f, p) \in V$ is a tuple indicating that flow f is assigned to request r at priority p. From now on, we consider that $\mathcal{F}$ consists uniquely of the shortest flows between each pair s, d. Now, consider a solution set $X \subseteq V$ which holds a candidate setup to solve a given instance of Problem 1. We denote by V^r the partition of the ground set V related to assignments for request r, such that $V = \left(V^1, \dots, V^{|\mathcal{R}|}\right)$.

The translation of Problem 1 to an equivalent problem in the set-function framework is rather straightforward and we introduce its formulation in Problem 2.

Problem 2 (TDP - Set Formulation)**.**

$$\underset{X\subseteq V}{\text{maximize}} \quad \text{QOS}(X) \triangleq \frac{1}{|\mathcal{R}|} \sum_{v\in X} T^{r(v)}_{p(v)} \tag{9}$$

subject to

$$|V^r \cap X| = 1, \forall r \in \mathcal{R} \tag{10}$$

$$\sum_{v\in X} \mathbb{1}_{d(v)=h} D^{r(v)}_{p(v)} \le C_h, \forall h \in \mathcal{H} \tag{11}$$

$$\sum_{v\in X\cap V^r} \sum_{e\in f(v)} \alpha_e \left[\sum_{v'\in X} \mathbb{1}_{e\in f(v')} T^{r(v')}_{p(v')} \right] + \beta_e \le L^r, \forall r \in \mathcal{R} \tag{12}$$

The objective function is given by (9). We ensure in (10) that there is only one element in X for each request. The computational capacity constraints are presented in (11), where we represent the destination node of the flow associated to element v by $d(v)$. Finally, the latency requirement constraints are depicted in (12).

In what follows, we use function $\textsc{IsValid} : 2^V \to \{0,1\}$, to indicate whether a set X is a valid solution, in terms of constraints (10), (11), and (12), or not. We also define the *implementation cost* of a provision plan X as

$$W(X) \triangleq \frac{1}{2|\mathcal{H}|}\sum_{h\in\mathcal{H}} \frac{\sum_{v\in X} \mathbb{1}_{d(v)=h} D_{p(v)}^{r(v)}}{C_h} + \frac{1}{2|\mathcal{R}|}\sum_{r\in\mathcal{R}} \frac{\textsc{Latency}^r(X)}{L^r}, \tag{13}$$

where, for a valid solution X, the values of $W(X)$ are limited within the interval $[0,1]$. Finally, inspired by [24], we propose to measure flows' centrality by the sum of their composing edges' *betweenness*[3], i.e.,

$$\mathrm{B}_f \triangleq \sum_{e\in f} \frac{1}{|\mathcal{F}|} \sum_{f'\in\mathcal{F}} \mathbb{1}_{e\in f'} \tag{14}$$

As the rate (2) primarily affects the latency, choosing less central shortest flows provides solution candidates with less chances to violate latency constraints.

B. Greedy Replacement Algorithm

In our first approach, we propose a greedy replacement algorithm, which, at every iteration, replaces element $v \in X$ with another element $v' \in V \setminus X$, of the same request (i.e., $r(v) = r(v')$), that provides the largest cost-benefit, i.e., marginal performance gain over the implementation cost. We describe its detailed operation in Algorithm 1.

Algorithm 1 has a simple implementation and runs in $\mathcal{O}(|V|^2)$, because the ground set update in line (26) enforces that replaced elements are not reconsidered. Even though we can not provide a PTAS (as discussed in Section III), we provide optimality guarantees under special cases.

Proposition IV.1. If requests' latency is independent of other requests, i.e., $\forall e \in \mathcal{E}, \alpha_e = 0$, then a provision plan $X_{\textsc{Greedy}}$ provided by Algorithm 1 enjoys $(1-1/e)$ optimality guarantee.

The objective function (9) is a monotone, submodular function and, in this case, the latency constraints can be converted into an additional set of knapsack constraints. Then, we reduce the submodular multiple knapsack problem to Problem 2 and, consequently, it holds that Algorithm 1's results are at most $(1 - 1/e)$ far from the optimal [26]. We refer to Appendix B for a complete proof.

Proposition IV.2. If, $\forall r \in \mathcal{R}$, $\forall p \in \mathcal{P}$, (i) $T_p^r = pT$ and (ii) $D_p^r = pD$, for some $T, D \in \mathbb{R}_+$, then a provision plan $X_{\textsc{Greedy}}$ provided by Algorithm 1 is optimal.

[3]Edge Betweenness [25] is a centrality in networks which assigns a score to an edge based on the fraction of shortest paths traversing it.

Algorithm 1: $\textsc{Greedy}$

```
input : $G = (\mathcal{H},\mathcal{E})$, $\mathcal{F}$, $\mathcal{R}$, $\mathcal{P}$, $V = (V^1,\ldots,V^{|\mathcal{R}|})$,
        Functions $\textsc{QoS}(\cdot)$, $W(\cdot)$, $\textsc{IsValid}(\cdot)$, and
        Parameters $\mathbf{D}$, $\mathbf{C}$, $\mathbf{T}$, $\mathbf{L}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{B}$.
output: Valid Provision Plan $X$
1  $X \leftarrow \emptyset$
2  repeat
3      $v_{\text{IN}} \leftarrow \textsc{None}$
4      $\Delta\,\textsc{QoS}^* \leftarrow 0$
5      $\Delta W^* \leftarrow 2$
6      for $v \in V \setminus X$ do
7          $X_{\text{REP}} \leftarrow \left(X \setminus V^{r(v)}\right) \cup \{v\}$
8          if $\textsc{isValid}(X_{\text{REP}})$ then
9              $\Delta\,\textsc{QoS} \leftarrow \textsc{QoS}(X_{\text{REP}}) - \textsc{QoS}(X)$
10             $\Delta W \leftarrow B_{f(v)}(W(X_{\text{REP}}) - W(X) + 1)$
11             if $\frac{\Delta\,\textsc{QoS}}{\Delta W} > \frac{\Delta\,\textsc{QoS}^*}{\Delta W^*}$ then
12                 $v_{\text{IN}} \leftarrow v$
13                 $\Delta\,\textsc{QoS}^* \leftarrow \Delta\,\textsc{QoS}$
14                 $\Delta W^* \leftarrow \Delta W$
15             else if $\Delta\,\textsc{QoS} = \Delta\,\textsc{QoS}^* = 0$ then
16                 if $\Delta W < \Delta W^*$ then
17                     $v_{\text{IN}} \leftarrow v$
18                     $\Delta\,\textsc{QoS}^* \leftarrow \Delta\,\textsc{QoS}$
19                     $\Delta W^* \leftarrow \Delta W$
20                 end
21             end
22         end
23     end
24     $v_{\text{OUT}} \leftarrow X \cap V^{r(v_{\text{IN}})}$
25     $X \leftarrow X \setminus \{v_{\text{OUT}}\} \cup \{v_{\text{IN}}\}$
26     $V \leftarrow V \setminus \{v_{\text{OUT}}\}$
27 until $\Delta\,\textsc{QoS}^* = 0$ and $\Delta W^* = 2$
28 return $X$
```

Due to space limitations, we present the detailed proof of Proposition IV.2 in Appendix C. In summary, the proof is based on the fact that the objective function of a solution X is simply given by the total assigned priority, i.e.,

$$\textsc{QoS}(X) = \frac{1}{|\mathcal{R}|}\sum_{v\in X} T_{p(v)}^{r(v)} = \frac{T}{|\mathcal{R}|}\sum_{v\in X} p(v) = \frac{T}{|\mathcal{R}|}P(X), \tag{15}$$

where $P(X)$ is solution X's priority level, i.e., the sum of the assigned priorities. This provides an instance of Problem 2 with optimal sub-structure, so we can gradually build the final solution by moving from a current solution X, in a priority level $P(X)$, to a new solution X', in an immediately higher priority level $P(X') = P(X) + 1$, until it reaches the optimal level $P(X_{\textsc{Greedy}}) = P(X^*)$.

C. Streaming Replacement Algorithm

We propose to solve the TDP using an algorithm based on the streaming framework, which consists in finding a solution with a single pass over the input set (see [18] for a thorough explanation). It evaluates each element in V and replaces the element of the same request in the current solution set X if it (i) provides a better QoS or (ii) provides equal QoS but at a smaller implementation cost. We describe the detailed procedure in Algorithm 2.

Algorithm 2: STREAMING

```
input : $G = (\mathcal{H}, \mathcal{E})$, $\mathcal{F}$, $\mathcal{R}$, $\mathcal{P}$, $V = (V^1, \ldots, V^{|\mathcal{R}|})$,
        Functions $\text{QoS}(\cdot)$, $W(\cdot)$, $\text{IsValid}(\cdot)$, and
        Parameters $\mathbf{D}$, $\mathbf{C}$, $\mathbf{T}$, $\mathbf{L}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{B}$.
output: Valid Provision Plan $X$
1  $X \leftarrow \emptyset$
2  for $v \in V$ do
3      $X_{\text{REP}} \leftarrow X \setminus V^{r(v)} \cup \{v\}$
4      if $\text{IsValid}(X_{\text{REP}})$ then
5          if $\text{QoS}(X_{\text{REP}}) > \text{QoS}(X)$ then
6              $X \leftarrow X_{\text{REP}}$
7          else if $\text{QoS}(X_{\text{REP}}) = \text{QoS}(X)$ then
8              if $W(X_{\text{REP}}) < W(X)$ then
9                  $X \leftarrow X_{\text{REP}}$
10             end
11         end
12     end
13 end
14 return $X$
```

Remark IV.1. The immediate replacement may quickly exhaust the available resources, stopping the algorithm from achieving better results. Let the *individual cost* of $v \in V$ be defined as

$$w_v \triangleq D^{r(v)}_{p(v)} + |f(v)|\, T^{r(v)}_{p(v)}.$$

In order to improve Algorithm 2's performance, we propose a 2-pass variant: Before effectively running the algorithm, we sort the elements in ascending order of individual cost.

V. Experimental Results

In this section, we first study the performance of our algorithms for different number of requests and network sizes. Then, we study how parameters α and β affect the optimal solution and the performance of the proposed approximate techniques. In our experiments, we consider a random MEC network consisting of 10 BSs and 20 NENs. We control the density of edges in the network with parameter $\rho \in (0, 1]$, such that $\rho = 1$ means a fully connected network. BSs and NENs have $C_h = 32.0$ GB and $C_h = 64.0$ GB of RAM, respectively. We consider 3 priorities, such that, $\forall r \in \mathcal{R}$, computational demands are $D^r_1 = 1.0$ GB, $D^r_2 = 2.0$ GB, and $D^r_3 = 4.0$ GB, and the throughput levels are $T^r_1 = 10.0$ Mbps, $T^r_2 = 20.0$ Mbps, and $T^r_3 = 30.0$ Mbps. Requests source hosts are chosen uniformly at random among the BSs and the related latency is also uniformly selected from the interval $[50.0, 150.0]$ ms. All these values are consistent with the literature (e.g., [15]).

Given the described experimental setup, we generate the request set such that BSs can provide all their UEs' requests at

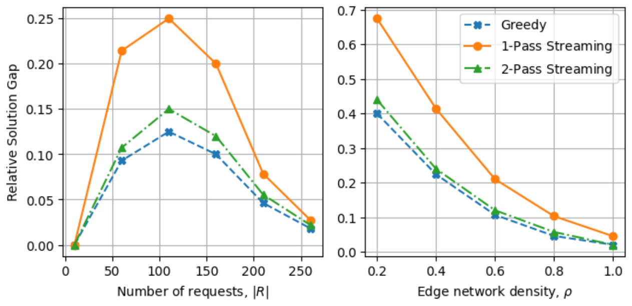

Fig. 1. Relative gap (left) versus the number of request $|\mathcal{R}|$ for $\rho = 0.6$ and (right) versus network density ρ for $|\mathcal{R}| = 110$.

the lowest priority possible. This guarantees that the optimization problem always have a feasible solution, which we refer to as the *trivial solution*. We propose to compare the performance of Algorithm 1 and Algorithm 2 single- and double-pass with Problem 1's upper bound (UB), provided by its continuous relaxation.

In the first set of experiments, we show the solution gap relative to the UB versus the number of requests $|\mathcal{R}|$, in Figure 1 (left), and versus the density ρ of the MEC network, in Figure 1 (right). In Figure 1 (left), we observe that all techniques start at zero, which means that the network can accommodate all requests in the highest priority. They all lose performance as we increase the number of requests and reach a peak in the solution gap at $|\mathcal{R}| = 100$. Interestingly, after that point, both curves seem to decrease asymptotically converging to zero. We assign this trend to the fact that, at some point, we reach the maximum number of requests the network can prioritize. Then, no matter how many additional requests we include, they will all get the lowest priority. For bigger request sets, the contribution of the excess of minimum-priority requests dominates the performance gain of the optimally assigned requests, making the relative solution gap approaching zero. In Figure 1 (right), we observe that the relative solution gap has its maximum value at $\rho = 0.2$, which means that flows tend to overshare the network edges, resulting in a solution close to the trivial one. The relative solution gap reduces as ρ increases, until it reaches $\rho = 1$, where the network is a complete graph and the flows are simply direct links. In this case, the problem's bottleneck is only the computational capacity of the hosts.

Now, consider that $\alpha_e = \alpha, \forall e \in \mathcal{E}$ and $\beta_e = \beta, \forall e \in \mathcal{E}$. In the second set of experiments, we show the relative solution gap versus parameters α, in Figure 2 (left), and versus the density β, in Figure 1 (right). In Figure 1 (left), we observe that, if $\alpha = 0$, the rate does not affect the latency and, thus, the problem is reduced to the computational constraints (in this setup, the hosts can provide all requests at the highest priority). The solution gap for the algorithms reach their peaks at $\alpha = 1$ and converge to zero, where only the trivial solution is feasible. Figure 1 (right) shows an abrupt decrease towards zero where only the trivial solution is feasible. Note that STREAMING had

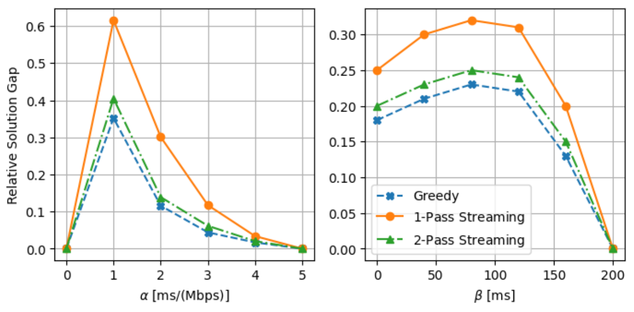

Fig. 2. Relative gap (left) versus the number of request α for $\beta = 80$ and (right) versus β for $\alpha = 1.5$.

its performance increased significantly by considering a pre-processing step, with results closer to the GREEDY's ones.

VI. Conclusions and Future Work

The TDP captures the primary requirements of latency-sensitive applications on top of MEC architectures. In this paper, we propose to approach the TDP with a flow-based optimization model. We also propose to approximate the latency through a linear model based on the classic end-to-end queuing delay. We approach the problem using two different replacement algorithms. The first one is based on greedy updates and, even though it is more computationally costly, it enjoys optimality guarantees under specific assumptions (e.g., see Proposition IV.2). The second one is designed within the streaming framework, which is able to quickly provide a feasible solution. In our experiments, we could observe our techniques' performance. Our proposed streaming algorithm is simple and, if considering two passes, may provide solutions with similar results to the greedy one. This framework can be expanded to consider a mathematical modeling for the robust optimization of the expected QoS under random requests and accounting for uncertainties.

References

[1] C. De Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang, and M. Liyanage, "Survey on 6g frontiers: Trends, applications, requirements, technologies and future research," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 836–886, 2021.

[2] CISCO, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper," CISCO, Tech. Rep., Feb 2017.

[3] Y. Zhang, "Mobile edge computing for beyond 5g and 6g," in *Mobile Edge Computing*. Springer, 2022, pp. 37–45.

[4] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.

[5] V. Stavroulaki, E. C. Strinati, F. Carrez, Y. Carlinet, M. Maman, D. Draskovic, D. Ribar, A. Lallet, K. Mößner, M. Tosic, M. Uitto, S. A. Hadiwardoyo x, J. Marquez-Barja, E. Garrido, M. Stamatelatos, K. Sarayeddine, P. Sánchez Vivas, A. Mämmelä, and P. Demestichas, "Dedicat 6g - dynamic coverage extension and distributed intelligence for human centric applications with assured security, privacy and trust: from 5g to 6g," in *2021 Joint European Conference on Networks and Communications & 6G Summit*, 2021, pp. 556–561.

[6] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, and R. Casado-Vara, "A review of edge computing reference architectures and a new global edge proposal," *Future Generation Computer Systems*, vol. 99, pp. 278–294, 2019.

[7] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[8] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[9] P. Wang, C. Yao, Z. Zheng, G. Sun, and L. Song, "Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2872–2884, 2019.

[10] Y. Jiao, P. Wang, D. Niyato, and Z. Xiong, "Social welfare maximization auction in edge computing resource allocation for mobile blockchain," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[11] Y.-J. Ku, P.-H. Chiang, and S. Dey, "Quality of service optimization for vehicular edge computing with solar-powered road side units," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–10.

[12] P. G. V. Naranjo, M. Shojafar, A. Abraham, and E. Baccarelli, "A new stable election-based routing algorithm to preserve aliveness and energy in fog-supported wireless sensor networks," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 002 413–002 418.

[13] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE internet of things journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[14] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[15] F. Spinelli, A. Bazco-Nogueras, and V. Mancuso, "Edge gaming: A greening perspective," *Computer Communications*, vol. 192, pp. 89–105, 2022.

[16] A. Benhamiche, A. Mahjoub, N. Perrot, and E. Uchoa, "Capacitated multi-layer network design with unsplittable demands: Polyhedra and branch-and-cut," *Discrete Optimization*, vol. 35, p. 100555, 2020.

[17] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–147, 1999.

[18] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: Massive data summarization on the fly," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 671–680. [Online]. Available: https://doi.org/10.1145/2623330.2623637

[19] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Algorithmic Aspects of Cloud Computing*, Y. Disser and V. S. Verykios, Eds. Cham: Springer International Publishing, 2019, pp. 73–101.

[20] M. Taha, L. Garcia, J. M. Jimenez, and J. Lloret, "Sdn-based throughput allocation in wireless networks for heterogeneous adaptive video streaming applications," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 963–968.

[21] D. Bertsekas and R. Gallager, *Data networks*. Athena Scientific, 2021.

[22] P. Bonami, D. Mazauric, and Y. Vaxes, "Maximum flow under proportional delay constraint," *Theoretical Computer Science*, vol. 689, pp. 58–66, 2017.

[23] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[24] Y. Zhuo, Y. Liang, Y. Huang, Y. Cao, J. Nie, and Y. Qi, "A routing strategy based on the betweenness centrality for multi-layers complex networks," in *2021 IEEE 9th International Conference on Information, Communication and Networks (ICICN)*, 2021, pp. 384–388.

[25] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008.

[26] X. Sun, J. Zhang, and Z. Zhang, "A tight deterministic algorithm for the submodular multiple knapsack problem," *arXiv preprint arXiv:2003.11450*, 2020.

Appendix A
Proof of Proposition III.1

Proposition III.1: Problem 1 is $\mathcal{NP}$-Hard.

Proof. Consider the Multiple Knapsack Problem (MKP): Let V be the ground set and K be the set of knapsacks. Each knapsack $k \in K$ has a capacity C_k and each $v \in V$ has a different cost $w_k(v)$ for each one of the knapsacks in K. Given a solution candidate $X \subseteq V$, the MKP consists in maximizing an arbitrary profit function $g : 2^V \to \mathbb{R}_+$ while satisfying the knapsack's capacity constraints, i.e.,

$$\sum_{v\in X} w_k(v) \leq C_k, \forall k \in K.$$

Note that the MKP is $\mathcal{NP}$-Hard (see, for example, [23]).

Now, consider a restricted form of the TDP, where latency does not depend on the rate, i.e., $\alpha_e = 0, \forall e \in \mathcal{E}$. As a consequence, we obtain that

$$\sum_{v\in X} B^i(v) \leq L^i, \forall i \in \mathcal{R},$$

where

$$B^i(v) = \begin{cases} \sum_{e\in f(v)} \beta_e, & \text{if } r(v) = i \\ 0, & \text{otherwise} \end{cases}.$$

We can also re-write the singularity constraints as

$$\sum_{v\in X} \mathbb{1}_{r(v)=j} \leq 1, \forall j \in \mathcal{R},$$

where we can change it to an inequality because TDP is a maximization problem, so it will always take one element for each request. Finally, the computational capacity constraints are unaffected by the initial condition and remains

$$\sum_{v\in X} \hat{D}_l(v) \leq C_l, \forall l \in \mathcal{H}$$

where

$$\hat{D}_l(v) = \begin{cases} D^{r(v)}_{p(v)}, & \text{if } d(v) = l \\ 0, & \text{otherwise} \end{cases}.$$

Note that we can summarize the entire set of constraints as

$$\sum_{v\in X} w_k(v) \leq \hat{C}_k, \forall k \in \mathcal{R} \cup \mathcal{R}' \cup \mathcal{H},$$

where $\mathcal{R}'$ is simply a copy of the original request set, the costs are defined as

$$\hat{w}_k(v) = \begin{cases} B^k(v) & \text{if } k \in \mathcal{R} \\ \mathbb{1}_{r(v)=k}, & \text{if } k \in \mathcal{R}' \\ \hat{D}_k(v), & \text{if } k \in \mathcal{H} \end{cases},$$

and the capacities are

$$\hat{C}_k = \begin{cases} L^r & \text{if } k \in \mathcal{R} \\ 1, & \text{if } k \in \mathcal{R}' \\ C_k, & \text{if } k \in \mathcal{H} \end{cases}.$$

Then, the MKP can be reduced to the described restricted form of the TDP. Therefore, the TDP is $\mathcal{NP}$-Hard as well. □

Appendix B
Proof of Proposition IV.1

Proposition IV.1: If requests' latency is independent of other requests, i.e., $\forall e \in \mathcal{E}, \alpha_e = 0$, then a provision plan X_{GREEDY} provided by Algorithm 1 enjoys $(1-1/e)$ optimality guarantee.

Proof. Similarly to what was done in Appendix A, if we consider $\forall e \in \mathcal{E}, \alpha_e = 0$, then we can represent the TDP as the following optimization problem:

$$\begin{aligned} &\underset{X\subseteq V}{\text{maximize}} \quad \text{QoS}(X) \\ &\text{subject to} \\ &\sum_{v\in X} \mathbb{1}_{r(v)=i} \leq 1, && \forall i \in \mathcal{R} \\ &\sum_{v\in X} \mathbb{1}_{d(v)=j} D^{r(v)}_{p(v)} \leq C_j, && \forall j \in \mathcal{H} \\ &\sum_{v\in X} \mathbb{1}_{r(v)=k} \sum_{e\in f(v)} \beta_e \leq L^k, && \forall k \in \mathcal{R} \end{aligned}$$

As already remarked, the problem above is equivalent to the MKP with three different set of knapsack constraints. We further note that the objective function is (i) monotone and (ii) submodular.

A set function $f : 2^V \to \mathbb{R}_+$ is monotone if, for all $X \subseteq Y \subseteq V$, $f(X) \leq f(Y)$. This is clearly the case for QoS because if you add a new element $v \in V \setminus X$ to the solution X, then we cannot decrease its value, i.e.,

$$\text{QoS}(X) \leq \text{QoS}(X \cup \{v\}), \quad \forall X \subseteq V, \forall v \in V \setminus X.$$

Therefore, QoS is monotone.

A set function $f : 2^V \to \mathbb{R}_+$ is submodular if, for all $X \subseteq Y \subset V$ and for some $v \in V \setminus Y$, $\Delta_f(X|v) \geq \Delta_f(Y|v)$, where $\Delta_f(X|v) \triangleq f(X \cup \{v\}) - f(X)$. Submodularity characterizes the "diminishing returns" property, which captures the idea that the benefit of adding elements to the solution is always larger in early-stage solutions. This is the case for the QoS function because

$$\begin{aligned} \Delta_{\text{QoS}}(X|v) &= \text{QoS}(X \cup \{v\}) - \text{QoS}(X) \\ &= \text{QoS}(X) + T^{r(v)}_{p(v)} - \text{QoS}(X) \\ &= T^{r(v)}_{p(v)} \\ &= T^{r(v)}_{p(v)} + \Delta_{\text{QoS}}(Y|v) - \Delta_{\text{QoS}}(Y|v) \\ &= \text{QoS}(Y \cup \{v\}) - \text{QoS}(Y) \\ &= \Delta_{\text{QoS}}(Y|v) \end{aligned}$$

In fact, the benefit of adding a new element $v \in V \setminus X$ to the solution X, i.e., $T^{r(v)}_{p(v)}$, is strictly equal during the entire solution process. Therefore, QoS is submodular.

We have then an instance of the Submodular Multiple Knapsack Problem (SMKP), i.e., the maximization of a monotone, submodular function subject to multiple knapsacks constraints. According to [26], for a problem with these characteristics, a greedy algorithm similar to Algorithm 1 is able to achieve a solution with $(1 - 1/e)$ approximation guarantee. □

Appendix C
Proof of Proposition IV.2

Consider the following assumptions: $\forall r \in \mathcal{R}$, $\forall p \in \mathcal{P}$

(i) $T_p^r = pT, T \in \mathbb{R}_+$ and
(ii) $D_p^r = pD, D \in \mathbb{R}_+$.

Firstly, we write the objective function as follows:

$$\text{QoS}(X) = \sum_{v \in X} T_{p(v)}^{r(v)} = \sum_{v \in X} p(v)\, T = K(X)\, T, \qquad (16)$$

where we define $K(X)$ to be the *priority level* of solution X.

In this case, the QoS of solutions admit discrete values depending on their priority levels. Note that, for any valid solution $X \subseteq V$, it has a priority level $K(X) \in \mathbb{Z}_+^*$, such that

$$|\,\mathcal{R}\,| \leq K(X) \leq |\,\mathcal{R}\,| \cdot |\,\mathcal{P}\,|.$$

Maximizing $\text{QoS}(X)$ is equivalent to maximize the priority level $K(X)$ and, from now on, instead of using function QoS, we will proceed with the proof simply in terms of $K(X)$.

At each priority level, solution candidates can be further grouped into different *assignment clusters*. Each assignment cluster gathers solutions with identical priority assignment but different used flows. We remark that, even though solutions in the same level have equivalent results, they may have different resource utilization patterns, and some of them even being unfeasible solutions.

Now, we define the replacement graph $\mathcal{G} = (2^V, E)$. The vertices of $\mathcal{G}$ are the solution candidates in the ground set V. Each node X has an edge to each other node X' differing from a pair of elements $v \in X, v' \in X'$ sharing the same request, i.e., if $X' = X \setminus \{v\} \cup \{v'\}$ and $r(v) = r(v')$, then $(X, X') \in E(\mathcal{G})$. We have two different types of edges:

1) Intra-cluster edge if $p(v) = p(v')$ (and $f(v) \neq f(v')$) and
2) Inter-cluster edge if $p(v) \neq p(v')$.

See the schematics presented in Figure 3 of a small example with $|\,\mathcal{R}\,| = |\,\mathcal{P}\,| = 3$.

We characterize the edges as follows:

- It is not possible to have an edge between nodes from different clusters at the same level.
- Each node is connected to all other nodes within the same cluster.
- If one node X from a cluster C at level k connects to one node X' from a cluster C' at level $k' > k$, then all nodes within C connect to all nodes within C'.
- A given cluster C at level k connects to exactly $|\,\mathcal{R}\,|$ other clusters at each other priority level $k' > k$.

In what follows, we provide a lemma that will help build the argument to support the claim in Proposition IV.2.

Lemma C.1. *It is always possible to reach the optimal solution X^* from the trivial solution X^0 and going through a path of feasible solutions on $\mathcal{G}$.*

Proof. The first step is to show that it is always possible to reach the optimal solution X^* from the initial solution X^0. Consider a generic feasible state X. In the base case, if there is an edge connecting X^0 and X, then there is a path $X^0 X$ of length 1. Otherwise, we can move to a solution X' in a lower priority cluster, i.e., $K(X') < K(X)$. It is guaranteed that there is a neighboring feasible X' because it necessarily has a smaller resource consumption. If we recursively apply the previous reasoning, we will eventually reach the base case, in which solution X'' is directly connected to X^0, with the resulting path $\{X, X', \ldots, X'', X^0\}$. Since $\mathcal{G}$ is undirected, the reverse path $\{X^0, X'', \ldots, X', X\}$ is also valid. Therefore, we conclude that there is a path from the initial state X^0 to any other feasible solution, including the optimal solution X^*. □

Proposition IV.2: If, $\forall r \in \mathcal{R}$, $\forall p \in \mathcal{P}$, (i) $T_p^r = pT$ and (ii) $D_p^r = pD$, for some $T, D \in \mathbb{R}_+$, then a provision plan X_{GREEDY} provided by Algorithm 1 is optimal.

Proof. Algorithm 1 starts with the empty solution and iteratively adds elements for each request. Note that, because we did not specify any visiting order, until all requests have an element, some replacements may already have happened. We start this proof from the point where all requests have already an assigned element. First, we will discuss the case where the initial solution is actually the trivial one, i.e., all BSs provide the requests for their UEs at the lowest priority possible. Then, we expand the idea and show that it is the case for any valid initial solution.

As we discussed in Lemma C.1, there is a path on $\mathcal{G}$ from the initial trivial solution X^0 to the optimal solution X^* through a sequence of valid solution. Let's call P_{GREEDY} the walk on graph $\mathcal{G}$ produced by the iterations of Algorithm 1.

The main characteristic of this walk is that we can only move from a solution in a priority level k to a priority level $k' \geq k$. This is the case because of lines (11) and (15) of Algorithm 1. Therefore, the replacements performed by Algorithm 1 cannot cause a reduction of priority level and, thus, of the objective function.

We proceed with a proof by contradiction: Assume it is not the case that Algorithm 1 provides an optimal solution. This means that the walk P_{GREEDY} starts at the initial solution X^0 and stops at final solution X^{GREEDY}, such that $K(X^{\text{GREEDY}}) < K(X^*)$. This would be the case only if it is not possible to replace any of the current elements in X^{GREEDY} such that you can either (A) improve the objective function or (B) reduce the implementation cost. However, since $K(X^*) > K(X^{\text{GREEDY}})$, there must be a solution X', such that $K(X') = K((X^{\text{GREEDY}})$ at the same level, i.e., $K(X^{\text{GREEDY}})$, but with a smaller implementation cost, i.e., $W(X') < W((X^{\text{GREEDY}})$, which will lead to a valid solution at a higher priority level (and further proceeding towards the optimal level $K(X^*)$). If such a solution exists, it must be located within a different cluster, otherwise Algorithm 1 would have been able to move to it by performing (B), i.e., reducing the implementation cost. There may be many different reasons why solution X' is able to advance to further levels and X^{GREEDY} was not. The question now is: Why Algorithm 1 would choose to go to X^{GREEDY} instead of X'? The answer is simple: it would not! And the reason is that since $K(X') = K((X^{\text{GREEDY}})$, the algorithm would necessarily choose the to move to the solution with the

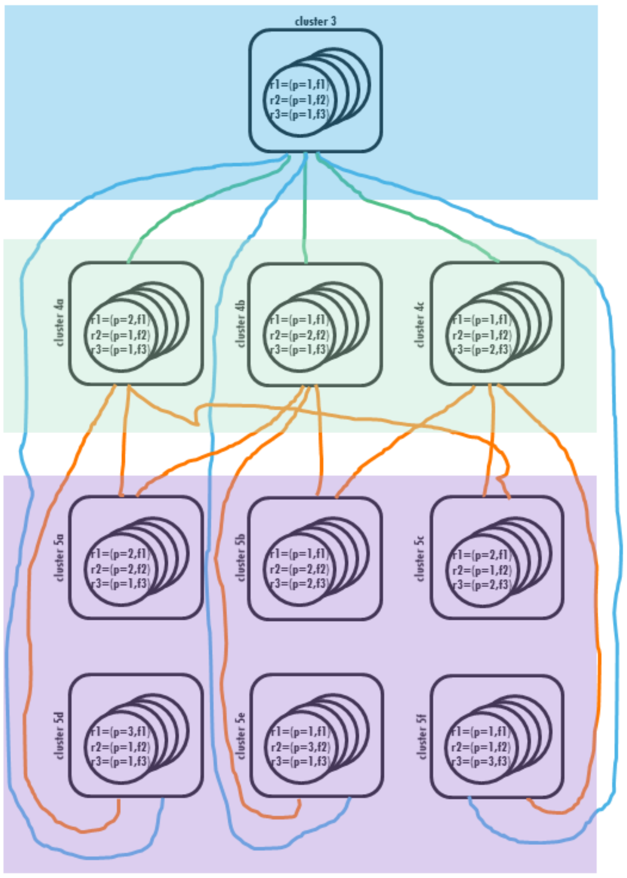

Fig. 3. Example of graph $\mathcal{G}$ for 3 requests and 3 priorities. Each shaded square represents a priority level.

smallest implementation cost (as it is enforced by line (11) of Algorithm 1). Then, if there is a solution with available resources to advance to further priority levels, Algorithm 1 would necessarily move to it. This reasoning can be applied to any solution in between and X^{GREEDY} and X^* and, in the end, the only possible explanation is that $K(X^*) = K((X^{\text{GREEDY}})$, which is a contradiction of our initial statement. Therefore, the walk produced by Algorithm 1 leads to an optimal solution.

$\square$