Multi-Objective Computation Sharing in Energy and Delay Constrained Mobile Edge Computing Environments

Arash Bozorgchenani, *Student Member, IEEE,* Farshad Mashhadi, *Student Member, IEEE,* Daniele Tarchi, *Senior Member, IEEE,* and Sergio Salinas, *Member, IEEE*

Abstract—In a mobile edge computing (MEC) network, mobile devices, also called edge clients, offload their computations to multiple edge servers that provide additional computing resources. Since the edge servers are placed at the network edge, e.g., cell-phone towers, transmission delays between edge servers and edge clients are shorter compared to those of cloud computing. In addition, edge clients can offload their tasks to other nearby edge clients with available computing resources by exploiting the Fog Computing (FC) paradigm. A major challenge in MEC and FC networks is to assign the tasks from edge clients to edge servers, as well as to other edge clients, in such a way that their tasks are completed with minimum energy consumption and minimum processing delay. In this paper, we model task offloading in MEC as a constrained multi-objective optimization problem (CMOP) that minimizes both the energy consumption and task processing delay of the mobile devices. To solve the CMOP, we design an evolutionary algorithm that can efficiently find a representative sample of the best trade-offs between energy consumption and task processing delay, i.e., the Pareto-optimal front. Compared to existing approaches for task offloading in MEC, we see that our approach finds offloading decisions with lower energy consumption and task processing delay.

Index Terms—Mobile Edge Computing, Fog Computing, Computation sharing, NSGA2, Multi-Objective Optimization, Evolutionary Algorithms, Energy Consumption, Delay.

1 Introduction

Mobile Edge Computing (MEC) is a promising computing platform that places computing resources near to the end-users, e.g., MEC servers can be co-located with cellular base stations. Compared to the existing Mobile Cloud Computing (MCC) infrastructure [1], [2], where computing resources are centralized in data centers far away from end-users, MEC offers a comparable amount of computing resources but with lower communication delays. To further increase the amount of computing resources available to end-users, it is possible to compliment MEC's edge resources with the computing capabilities of other end-users' devices, also known as fog computing [3], [4]. Due to their high potential, both MEC and Fog computing have been used to handle applications with low-latency and high-throughput requirements. For example, in Industry 4.0, manufacturers equip their workers with augmented reality headsets that require significant computing resources to render three-dimensional images [5], [6]. In smart healthcare, physiological data collected by Internet of Things (IoT) devices needs to be quickly analyzed to provide timely diagnoses [7], [8].

In MEC networks, resource-poor end-user devices, called Edge Clients (ECs), can reduce their computing time by offloading their computing tasks to nearby resource-rich edge servers, called Edge Nodes (ENs), that are able to complete the computing tasks faster than the ECs. Besides offloading computing tasks to ENs, ECs can also outsource their computations to other ECs that are willing to share their idle resources with them.

Although ENs can complete computing tasks faster than the ECs, offloading tasks to a remote device comes with additional energy consumption and time delays, which can, in some cases, exceed those of computing the task locally. In particular, when ECs offload their tasks, they must first spend energy to transmit their data to the remote device, then wait for the data to be uploaded, and, finally, spend energy for receiving the task results. Thus, the ECs face a trade-off between the overall task completion time and their energy consumption when deciding whether to offload their tasks to other devices.

To find optimal offloading decisions in MEC, we model the task offloading problem as a constrained multi-objective optimization problem (CMOP) that jointly minimizes the task processing delay and the energy consumption of the ECs. The solution to the CMOP is characterized by a Pareto front formed by a set of possible solutions where each solution represents a different trade-off between task processing delay and energy consumption of the ECs. To solve the CMOP, we propose an evolutionary algorithm (EA) that can efficiently find a high-quality approximation of the Pareto-optimal front. Specifically, we use the Non-dominated Sorting Genetic Algorithm 2 (NSGA2) [9], which has been shown to be fast and effective at solv-

- *A. Bozorgchenani and D. Tarchi are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, Italy.*
- *F. Mashhadi and S. Salinas are with the Department of Electrical Engineering and Computer Science, Wichita State University, USA.*

This work has been partially supported by the project "GAUChO - A Green Adaptive Fog Computing and Networking Architecture" funded by the MIUR Progetti di Ricerca di Rilevante Interesse Nazionale (PRIN) Bando 2015 - grant 2015YPXH4W_004.

ing constrained multi-objective problems such as the one considered in this paper. Different to existing EAs, which use a set of random solutions for initialization, our EA leverages knowledge about the ECs energy resources and location to generate an initial set of solutions that is closer to the Pareto-optimal front compared to a randomly generated one. Our extensive simulations show that using the proposed initialization technique allows our EA to find a high-quality approximation of the Pareto-optimal in fewer iterations than existing EAs. Moreover, we also analyze the results obtained by our algorithm when we set a maximum number of iterations, and under varying weights for the energy consumption and task processing delay.

The rest of the paper is organized in the following way. In Section 2, we review the literature. In Section 3, we describe the system. Section 4 describes the proposed EA to solve the MEC CMOP, and, in Section 5, we present our numerical results. Finally, in Section 6, we give concluding remarks.

2 Related Works

Task offloading has been widely investigated in cloud computing, MEC and fog computing. In this section, we briefly summarize the existing approaches for task offloading.

Devices that offload computing tasks to remote servers have two objectives. First, they aim to minimize the time it takes them to complete the computing task. Second, they aim to employ a minimum amount of energy to preserve their scarce energy resources. Some existing works exclusively focus on minimizing the task completion time. In [10], Yang et al. propose to minimize the task completion delay of a set of mobile devices that offload their tasks to the cloud. They also propose an offline heuristic to solve their optimization problem. Jia et al. [11] minimize the task completion delay in cloud computing while considering the finite energy resources at the mobile devices as optimization constraints. Ning et al. [12] minimize the task completion delay in a MEC network while ignoring task queues at the edge servers. Liu et al. [13] consider a MEC network and propose a task scheduling policy that minimizes the task computing delays of the users' devices. Chen et al. [14] minimize the task computing time by modeling the problem as a semi-Markovian decision process. Li et al. [15] propose an online mechanism where a base station selects the task executors of devices' requests by jointly optimizing resource allocation and time scheduling.

There are some works that focus on minimizing the energy consumption of the mobile devices. Specifically, Chen et al. [16] use an online peer offloading framework based on Lyapunov optimization to maximize the long-term network performance while keeping low the energy consumption of small-cell base stations in a MEC network. Zhang et al. [17] develop a scheme that optimally chooses between local computation or offloading to the cloud to minimize the energy consumption. In [18], You et al. propose optimal cloud outsourcing mechanisms for mobile devices capable of transferring and harvesting power. The authors in [19] formulate the problem of minimizing energy consumption for a collaborative task execution between one mobile device and one cloud infrastructure as a constrained shortest path problem. In [20], Mahmoodi et al. propose an optimal computation offloading schedule of a mobile application to the cloud subject to the saved energy at the mobile users. Cao et al. [21] present a computation and communication cooperation scheme for MEC systems. The authors considered multi-hop task offloading where intermediate devices between the offloading devices and the edge servers can participate in task execution. Song et al. [22] propose a pricing mechanism and a Lyapunov optimization scheme that can guarantees fair energy consumption between mobile devices. The authors in [23] aim at minimizing the energy consumption for mobile devices in MEC respecting the power and latency constraints. After decomposing the problem, they propose an iterative solution to solve transmission power allocation and computation offloading. However, we see that the above works only minimize the energy consumption while ignoring the the task completion delay.

As introduced in Section 1, there is a trade-off between task completion delay and energy consumption of the mobile devices. Some existing works have proposed task offloading schemes that aim to simultaneously optimize both objectives. Specifically, in [24], Kao et al. propose a task offloading algorithm that balances energy consumption costs and latency in latency-sensitive applications. Wang et al. [25] consider mobile device energy consumption, and application execution latency but formulate separate minimization problems. They first obtain an optimal solution for the energy consumption minimization problem and then a locally optimal solution for the latency minimization problem. Hong et al. [26] consider both energy consumption and latency by formulating an aggregate objective function. Jiang et al. [27] propose a Lyapunov optimization approach for cloud offloading scheduling, where multiple applications are running on multi-core CPU mobile devices. Dinh et al. [28] jointly minimize task execution latency and energy consumption of mobile devices by optimally choosing task offloading decisions and the CPU-cycle frequency of the mobile devices. In [29], Wang et al. propose a framework that optimally offloads computation offloading and assigns wireless resources. The MEC server first makes the offloading decision based on the estimated overhead for mobile devices and itself. Then, by solving a graph coloring problem, the framework assigns wireless channels. The authors in [30] propose an energy-aware offloading approach that minimizes the weighted sum of energy consumption and latency by optimally choosing the computation offloading decision, local computation frequency scheduling, and allocation of power and wireless channels. In [31], Cao et al. consider a simple three-nodes MEC architecture formed by a user node, a helper node, and an access point. They propose to divide the task into three parts and assign each part to one of the nodes. The objective is to jointly Chen et al. in [32], propose a gradient-based method for energy minimization while meeting certain values of delay and energy consumption. Lu et al. [33] address an edge-enabled IoT scenario and study computation offloading by considering task latency, energy consumption, and task success rate through the use of a deep reinforcement algorithm. The work in [34] studies a fog-based mobile cloud computing, where mobile devices are modeled with queues, fog nodes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.2994232, IEEE Transactions on Mobile Computing

act as access points, and a central cloud is available for computations. The authors model the energy consumption and task delay offloading model as a multi-objective optimization that minimizes energy consumption, latency and cloud payments. To solve this problem, the authors relax the multi-objective optimization into a single-objective problem and solve it using the interior point method. Although these works aim to simultaneously optimize task completion delay and mobile device energy consumption, their proposed solution methods form a single objective, e.g., [34], or finally consider the objectives separately, which yields only one trade-off between the two objectives and ignores a large part of the trade-off space.

To better explore the trade-off space in multi-objective optimization problems, researchers have considered metaheuristic solution approaches. In [35], Midya et al. propose an algorithm that combines a genetic algorithm (GA) with adaptive particle swarm optimization to offload tasks in a vehicular cloud. Cui et al. [36] employ an evolutionary algorithm to minimize energy consumption and task completion delay in a MEC network where mobile devices offload their tasks to the edge servers. Unfortunately, these works only consider mobile devices that offload their tasks to either the cloud or edge servers while ignoring the possibility of offloading tasks to other mobile devices with idle computing and energy resources.

In this paper, we propose an evolutionary algorithm that can simultaneously minimize task completion delays and energy consumption while considering task offloading to both edge servers, i.e., ENs, and to other mobile devices, i.e., ECs. Compared to previous works, our approach efficiently explores the trade-off space and yields a set of different solutions with different trade-offs between the two objectives. Besides, our approach scales well in the number of devices in the MEC network and can handle partial task offloading, which results in fine-grained control of the offloading process and lower completion delays and energy consumption.

3 System Model and Problem Formulation

We consider a two-tier MEC architecture formed by a set $\mathcal{U} = \{u_1, \ldots, u_i, \ldots, u_N\}$ of heterogeneous ECs, with limited computational capabilities and acting as sources of the computation requests, and a set of ENs $\mathcal{F} = \{f_1, \ldots, f_m, \ldots, f_M\}$, characterized by a higher computation capability. The heterogeneous ECs have different computational capabilities and energy requirements. ECs are battery powered, and, thus, have limited energy resources, while ENs are connected to the electrical network, and have access to virtually unlimited energy. A system operator located nearby to the operational scenario manages the MEC network by collecting information about the ECs and ENs and estimating the offloading amounts that minimize the energy consumption and task processing delay of the ECs.

To reduce both task completion delay and energy consumption, ECs can offload their computational tasks through wireless links to other nearby ECs or ENs. Allowing task offloading between ECs is particularly important to meet the task completion delay requirements of low latency communications in battery operated edge environments.

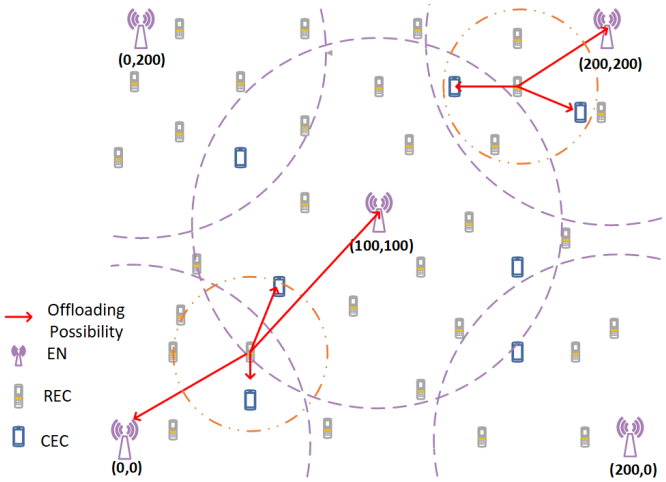


Fig. 1. An Architecture for Task Offloading in Mobile Edge Computing (MEC)

Thus, we assume that each EC can split its tasks into multiple unique portions and simultaneously offload each portion to different ECs or ENs. Fig. 1 shows the considered MEC architecture. For a better clarity of the system to be presented in the following, we list the key notations in Table 1.

Consider an EC u_i having a computing task that needs to be processed, and its set of neighboring devices, which is given by

$$\begin{aligned}\mathcal{N}(i) &= \mathcal{N}_{EC}(i) \cup \mathcal{N}_{EN}(i) \\ &= \{u_j | d(u_i, u_j) \leq R_{\mathcal{U}}, \forall j\} \cup \{f_m | d(u_i, f_m) \leq R_{\mathcal{F}}, \forall m\} \\ &= \{\nu_1^i, \ldots, \nu_k^i, \ldots, \nu_{N_i}^i\} \end{aligned} \quad (1)$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance operator between devices, $R_{\mathcal{U}}$ is the coverage range of ECs, $R_{\mathcal{F}}$ is the coverage range of ENs, ν_k^i identifies the generic kth device belonging to the neighborhood of u_i, and N_i is the total number of neighbors of EC u_i. Note that the neighbor set cardinality is variable across the devices, and that a device can belong to different neighbor sets if it can be served by more than one device.

The task offloading operation can be performed in two ways. The first way is to offload the computing task from an EC as a whole to a single device. The second way is to first partition the EC's computing task into several segments that can be computed in parallel, and then offload each task to a different device. We call the second approach task partitioning. Since task partitioning offers more opportunities to utilize the idle computing resources in the network, we adopt this approach in our system model.

Each EC can offload a set of portions of a single task to another EC or EN. We assume that each set of offloaded portions can be individually outsourced and computed in parallel. For instance, in Fig. 2, EC u_i has one task with ten portions, and three available devices to offload, out of which two are other ECs and one is an EN. In this example, five fixed-size portions are offloaded to u_1, two are offloaded to u_2 and three are offloaded to f_1. The set of portions to be

TABLE 1
Nomenclature (in order of appearance)

Notation	Description
$\mathcal{N}_{EC}(i)$	Set of ECs within u_i's range.
$\mathcal{N}_{EN}(i)$	Set of ENs within u_i's range.
$R_{\mathcal{U}}$	Coverage range of ECs
$R_{\mathcal{F}}$	Coverage range of ENs
ν_k^i	kth device belonging to the neighborhood of u_i
δ_{tx}, δ_{rx}	Minimum transmitted and received task portion size
α	Ratio of δ_{tx} to δ_{rx}
γ_i	The number of portions in u_i's task
η_{c_j}	computational capability of device u_j
η_{c_m}	computational capability of device f_m
O_l^i	number of processing operations for lth task of u_i
$\phi_{u_i}^{u_j}$	Amount of task portions of u_i shared with u_j
$\phi_{f_m}^{u_j}$	Amount of task portions of u_i shared with f_m
E_{off}^{i,ν_k^i}	Task portion offloading energy consumption by u_i to ν_k^i
$\bar{E}_{off}^i$	Partial offloading energy consumption by u_i
T_{off}^{i,ν_k^i}	Task portion offloading delay by the u_i
T_{off}^i	Overall offloading delay by the u_i
$\bar{n}$	Number of ECs in the REC subset.
$\hat{n}$	number of CECs within reach of u_i.
$\hat{f}$	Number of ENs within range of u_i.
E_r^i	Remaining energy of EC u_i.
$F_{E_r}(i)$	Distribution of the remaining energy of all ECs.
K_i	Maximum number of devices that can accept offloaded tasks.
$\tilde{w}_{u_i}^{\nu_k^i}$	Sum of energy and delay for possible offloading decision of device u_i.
$\tilde{E}_{off}^{i,\nu_k^i}$	Normalized value of E_{off}^{i,ν_k^i} in range of [0,1].
$\tilde{T}_{off}^{i,\nu_k^i}$	Normalized value of T_{off}^{i,ν_k^i} in range of [0,1].
$\phi_{u_i}^{\nu_k^i}$	Initial offloading decision for device u_i.
$\mathbf{\Phi}_0$	The initial solution population.
Φ_z	Solution matrix, where $\Phi_z \in \mathbf{\Phi}_0$.
E_{Φ_z}	Overall energy consumption of solution Φ_z.
T_{Φ_z}	Overall processing delay of solution Φ_z.
I_z	Proximity of solution z with other solutions in the objective space.
W	Maximum number of crossover operations.
P_C	Probability of performing crossover operation.
P_M	Probability of performing mutation operation.
$\mathcal{Q}_t$	Generated offspring set.

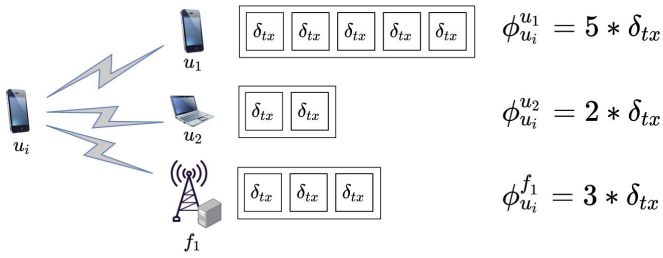


Fig. 2. Task portion distribution

estimated and shared from EC u_i to EC u_j and from EC u_i to EN f_m, is denoted as $\phi_{u_i}^{u_j}$ and $\phi_{u_i}^{f_m}$, respectively.[1]

To partition the tasks into meaningful portions, the portion sizes are set equal to a multiple of a minimum size. The minimum portion size is given by the smallest number of bytes that are needed to convey an instruction of the task's application, e.g., a given number of floating point operations (FLOPS). Moreover, we assume the task result size is smaller than size of the offloaded task. Let δ_{tx} be the minimum task portion size which can be transmitted, and δ_{rx} the minimum task portion size which can be received. Thus, the ratio between one offloaded portion and one downloaded portion is defined as $\alpha = \delta_{tx}/\delta_{rx}$, where $\alpha > 1$, and depends on the application type. Now, let D_s^i be the size of u_i's task to be offloaded, and D_r^i be the size of u_i's task to be downloaded. Then, the total number of portions in u_i's task is given by:

$$\gamma_i = \frac{D_s^i}{\delta_{tx}} = \frac{D_r^i}{\delta_{rx}} \tag{2}$$

where we assume that we can pad the task size to make D_s^i a multiple of δ_{tx} and, similarly, D_r^i a multiple of δ_{rx}. Let $\phi_{u_i}^{u_j}$'s and $\phi_{u_i}^{f_m}$'s be positive integers less than γ_i. For all the offloaded portions we have the following:

$$\sum_{j=1}^{N} \phi_{u_i}^{u_j} + \sum_{m=1}^{M} \phi_{u_i}^{f_m} = \gamma_i \tag{3}$$

To offload tasks, the ECs collaborate with other ECs and ENs in three phases. First, the originating EC transmits the task's data to a neighboring EC or EN. Second, the neighboring device completes the computational task. Third, the originating EC downloads the task result from the neighboring device. Thus, the originating EC's energy consumption for offloading a single task portion[2] of size δ_{tx} to a single neighboring device is:

$$E_{off}^{i,\nu_k^i} = E_{tx}^{i,\nu_k^i} + E_{rx}^{i,\nu_k^i}, \qquad \forall \nu_k^i \in \mathcal{N}(i) \tag{4}$$

where E_{tx}^{i,ν_k^i} and E_{rx}^{i,ν_k^i} are the energy spent for transmitting a single task portion and for receiving the results from remote device $\nu_k^i \in \mathcal{N}(i)$, respectively[3].

Now, the total energy consumption of the EC is given by

$$\bar{E}_{tot}^i = \sum_{u_j, f_m \in \mathcal{N}(i)} \left(\phi_{u_i}^{u_j} \left(E_{tx}^{i,u_j} + E_{rx}^{i,u_j} \right) + \phi_{u_j}^{u_i} \cdot E_{com}^i + \phi_{u_i}^{f_m} \left(E_{tx}^{i,f_m} + E_{rx}^{i,f_m} \right) \right) + E_{id}^i \tag{5}$$

where E_{com}^i represents the energy spent by the EC u_i to compute a single portion of a task, while E_{id}^i denotes the energy consumption of EC u_i when it is idle, i.e., neither transmitting, receiving or computing. We have considered $\phi_{u_i}^{u_i}$ as a special case when $u_j = u_i$, corresponding to the case in which the EC is offloading to itself i.e., processing locally its own task. In this case, $E_{tx}^{i,u_i} = 0$ and $E_{rx}^{i,u_i} = 0$.

Similarly, we model the offloading delay for a single portion as:

$$T_{off}^{i,\nu_k^i} = T_{tx}^{i,\nu_k^i} + T_{rx}^{i,\nu_k^i} + T_{com}^{i,\nu_k^i}, \qquad \forall \nu_k^i \in \mathcal{N}(i) \tag{6}$$

where T_{tx}^{i,ν_k^i} and T_{rx}^{i,ν_k^i} are the time needed to transmit a single task portion to a remote device and the time needed to

1. Since some of the neighbors of EC may offer a long task processing delay, or result in high energy consumption, the EC may decide to not outsource any task portion to some of its neighbors.

2. For a single portion, $\phi_{u_i}^{u_j}$ or $\phi_{u_i}^{f_m}$ equals to one. Note that the sum of these two parameters, which represent the total number of offloading portions, should be equal to γ_i as defined in (3).

3. Note that the reception energy is smaller than the transmission energy since $\delta_{tx} > \delta_{rx}$.

receive the results, respectively, and T_{com}^{i,ν_k^i} is the computation time of the task portion at the remote device ν_k^i. The time period T_{com}^{i,ν_k^i} is also the time that the EC u_i waits to, between uploading the task portion and receiving the result.

Given a task portion with size δ_{tx}, the transmission time is defined as:

$$T_{tx}^{i,\nu_k^i} = \frac{\delta_{tx}}{r_{i,\nu_k^i}} \tag{7}$$

and the corresponding receiving time as:

$$T_{rx}^{i,\nu_k^i} = \frac{\delta_{rx}}{r_{i,\nu_k^i}} \tag{8}$$

where r_{i,ν_k^i} is the data rate of the link between the EC u_i and the remote device ν_k^i in the set of neighboring devices $\mathcal{N}(i)$.

In this work, we consider that ECs generate tasks with variable size. We define the index of a single task as l. Hence, the time spent by device ν_k^i to compute one portion of the lth task produced by the EC u_i corresponds to:

$$T_{com}^{i,\nu_k^i} = \begin{cases} \frac{O_l^i/\gamma_i}{\eta_{c_j}}, & \text{if } \nu_k^i \in \mathcal{N}_{EC} \\ \frac{O_l^i/\gamma_i}{\eta_{c_m}}, & \text{if } \nu_k^i \in \mathcal{N}_{EN} \end{cases} \tag{9}$$

where O_l^i represents the number of processing operations related to the lth task produced by the EC u_i, and η_{c_j} and η_{c_m} are the Floating-point Operations Per Second (FLOPS) depending on the CPU of the EC u_j or the EN f_m, respectively. Thus, O_l^i/γ_i represents the number of processing operations for a single portion.

A remote device ν_k^i may receive multiple task portions from other ECs or ENs. Therefore, the arriving task portion of EC u_i at the remote device ν_k^i may experience a waiting time before it is processed. Thus, we can write the overall offloading time as follows:

$$\bar{T}_{off}^i = \max_{u_j, f_m \in \mathcal{N}(i)} \left(\phi_{u_i}^{u_j} \cdot T_{off}^{i,u_j} + T_w^{i,u_j}, \phi_{u_i}^{f_m} \cdot T_{off}^{i,f_m} + T_w^{i,f_m} \right) \tag{10}$$

where T_w^{i,u_j} is the task portion waiting time at the remote device due to other processes.

Our goal is to efficiently find the number of task portions that ECs should offload to their neighboring ECs, and to their neighboring ENs, i.e., $\phi_{u_i}^{u_j}$'s and $\phi_{u_i}^{f_m}$'s, respectively, that simultaneously minimize the EC's overall energy consumption and task processing delay.

3.1 A Constrained Multi-objective Optimization Problem for Task Offloading in Edge Computing

To define the task offloading problem, we observe that an EC aiming to minimize its task processing delay would attempt to offload task portions to as many neighboring devices as possible. The reason is that this would allow it to exploit the neighbors' computing resources in parallel. However, increasing the number of devices for offloading incurs in an increased energy consumption due to the additional energy needed for transmission and reception to devices that are further away. Indeed, in the considered task offloading operation, delay is minimized when

$$\max \sum_{k=1}^{|\mathcal{N}(i)|} \mathbb{1}\{\phi_{u_i}^{\nu_k^i} > 0\}$$

for every single task of u_i that we have. This drives to the observation that the EC's energy consumption and task processing delays are competing objectives.

To efficiently handle these two competing objectives, we formulate the task offloading problem in edge computing as a Constrained Multi-Objective Optimization Problem (CMOP) in (11), subject to the constraints:

$$Eq.\ (3), \tag{12a}$$
$$\phi_{u_i}^{u_j} = 0, \quad \text{if } u_j \notin \mathcal{N}(i), \tag{12b}$$
$$\phi_{u_i}^{f_m} = 0, \quad \text{if } f_m \notin \mathcal{N}(i), \tag{12c}$$
$$\phi_{u_i}^{u_j} \geq 0, \quad \text{if } u_j \in \mathcal{N}(i), \tag{12d}$$
$$\phi_{u_i}^{f_m} \geq 0, \quad \text{if } f_m \in \mathcal{N}(i), \tag{12e}$$

where Φ is the variable vector denoting the task offloading decisions for the ECs, i.e.,

$$\Phi = \begin{pmatrix} \phi_{u_1}^{u_1} & \cdots & \phi_{u_1}^{u_j} & \cdots & \phi_{u_1}^{u_N} & \phi_{u_1}^{f_1} & \cdots & \phi_{u_1}^{f_m} & \cdots & \phi_{u_1}^{f_M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{u_i}^{u_1} & \cdots & \phi_{u_i}^{u_j} & \cdots & \phi_{u_i}^{u_N} & \phi_{u_i}^{f_1} & \cdots & \phi_{u_i}^{f_m} & \cdots & \phi_{u_i}^{f_M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{u_N}^{u_1} & \cdots & \phi_{u_N}^{u_j} & \cdots & \phi_{u_N}^{u_N} & \phi_{u_N}^{f_1} & \cdots & \phi_{u_N}^{f_m} & \cdots & \phi_{u_N}^{f_M}. \end{pmatrix} \tag{13}$$

In (11), the first objective minimizes the EC's energy consumption, and the second objective minimizes the task processing delay for all ECs. Constraint (12a) guarantees that all task portions are computed either locally or at a remote device, as already defined in (3). Constraints (12b), and (12c) prevent the ECs from offloading tasks to non-neighboring devices, and (12d) and (12e) constraints the offloading decisions to non-negative values.

3.2 Characterization of the Pareto-optimal Front for Task Offloading CMOP in Edge Computing

Unlike single-objective optimization problems where there is a unique solution, the task offloading CMOP in edge computing is characterized by a Pareto front of solutions.

In the presence of multiple conflicting objectives, any solution point has to be gauged along multiple dimensions. Hence, the quality of a solution is determined by its Pareto-dominance with respect to other solutions. In particular, let $\mathbf{\Phi} = \{\Phi_1, \Phi_2, ..., \Phi_Z\}$ be the set of solutions, where Φ_z is the zth solution as represented in (13), and Z is the total number of generated solutions. Considering two solutions, say Φ_1 and Φ_2, for a given problem with C conflicting objectives, say ω_c (for all $c \in [1, C]$), we define Pareto-dominance as follows:

Definition 1. *Let $\omega_c(\Phi)$ be the value of the objective function for the cth objective evaluated at some solution Φ. Then Φ_1 is said to Pareto-dominate Φ_2 (i.e., $\Phi_1 \succ \Phi_2$) if $\omega_c(\Phi_1) \leq \omega_c(\Phi_2)$ for all $c \in [1, C]$, and there exists some $p \in [1, C]$ such that $\omega_p(\Phi_1) < \omega_p(\Phi_2)$.*

Although the above Pareto-dominance definition allows to classify solutions based on their quality, it treats feasible and unfeasible solutions equally. To favor feasible solutions and penalize those that violate the constraints, we adopt the constraint-dominance definition proposed in [9], i.e.,

Definition 2. *A solution vector Φ_1 is said to constraint-dominate another solution vector Φ_2 if any of the following conditions is true:*

$$\min_{\Phi}\Bigg\{\sum_{i=1}^{N}\Bigg(\sum_{u_j,f_m\in\mathcal{N}(i)}\left(\phi_{u_i}^{u_j}\left(E_{tx}^{i,u_j}+E_{rx}^{i,u_j}\right)+\phi_{u_j}^{u_i}\cdot E_{com}^{i}+\phi_{u_i}^{f_m}\left(E_{tx}^{i,f_m}+E_{rx}^{i,f_m}\right)\right)\Bigg),$$
$$\sum_{i=1}^{N}\left(\max_{u_j,f_m\in\mathcal{N}(i)}\left\{\phi_{u_i}^{u_j}\left(\frac{\delta_{tx}}{r_{i,u_j}}+\frac{\delta_{rx}}{r_{i,u_j}}+\frac{O_l^i/\gamma_i}{\eta_{c_j}}\right)+T_w^{i,u_j},\phi_{u_i}^{f_m}\left(\frac{\delta_{tx}}{r_{i,f_m}}+\frac{\delta_{rx}}{r_{i,f_m}}+\frac{O_l^i/\gamma_i}{\eta_{c_m}}\right)+T_w^{i,f_m}\right\}\right)\Bigg\} \quad (11)$$

1) *Φ_1 is feasible, i.e., it satisfies all constraints, but Φ_2 is not.*
2) *Both Φ_1 and Φ_2 are feasible and Φ_1 Pareto-dominates Φ_2.*
3) *Both Φ_1 and Φ_2 are infeasible, but Φ_1 has lower overall constraint violation.*

Consequently, we can define the set of non-dominated solutions as:

$$S^P=\{\Phi_a \mid \nexists\Phi_b\succ\Phi_a,\ \text{for } 1\le a,b\le Z\} \quad (14)$$

In the following section, we design an evolutionary algorithm that can find high-quality approximations of the solution to the CMOP in (11).

4 An Evolutionary Algorithm for Task Offloading in Edge Computing

To solve the task offloading CMOP described in Section 3.1, we propose a Multi-Objective Evolutionary Algorithm (MOEA). The main idea of a MOEA is to use the evolutionary principles of crossover, mutation, and selection of Darwinian evolution to find the Pareto front. Crossover and mutation probabilistically combine solutions to find possibly better new solutions, while selection deterministically discards low-quality solutions and keeps high-quality ones. Compared to other methods for multi-objective optimization, MOEAs offer an efficient way to find a high-quality approximation of the Pareto-optimal front in a single run.

Specifically, our proposed MOEA operates in four steps: initialization, selection, reproduction, and population update. First, the proposed algorithm randomly generates an initial solution population, and ranks these solutions based on their quality using selection. In the reproduction step, the proposed algorithm probabilistically combines the high-quality solutions with each other to generate possibly better new ones. Then, the proposed algorithm repeats the selection and reproduction steps until it reaches a maximum number of iterations. In the following, we explain these steps in details.

4.1 Initialization

To reduce the number of iterations needed to find a high-quality approximation of the Pareto-optimal front, we develop a two-phase initialization procedure that leverages the structure of the task offloading problem to find a high-quality initial solutions set. The proposed method generates initial solutions with better quality compared to the ones generated by random initialization, which allows us to find a high-quality approximation of the Pareto-optimal front in a low number of iterations. We measure the quality of our initial solutions in the numerical result section.

In the first phase, we discard the devices that have low energy levels. The reason is that selecting these devices to compute the tasks from the ECs could deplete their energy. In addition, discarding low energy devices significantly reduces the solution space. In the second phase, we propose a weighted random solution generation that favors offloading decisions that delegate tasks between nearby devices, resulting in initial offloading decisions with lower energy consumption and task processing delay.

4.1.1 Phase 1: EC Classification

We divide EC's into two subsets depending on their energy level. The first subset contains ECs that have enough energy to perform the computing tasks on behalf of other ECs. We call this subset the Computing EC's (CEC) subset. The second subset contains EC's that lack enough energy to perform their own computations, and thus offload their tasks to other EC's. We call this subset the Requesting EC's (REC) subset. In order to perform this classification there could be several methods aiming at sorting the devices and dividing the set into two subsets. In this paper, we classify the ECs based on the distribution of their remaining energy at the moment the classification is performed. This approach allows us to take into account the actual energy level of the ECs. In particular, the two sets are selected based on a 3-quantile function, so that they are always balanced in numerosity [37]. To this aim, we formally define the CEC and REC subsets as follows:

$$\mathcal{U}_{REC}=\left\{u_i|E_r^i\le E_{r,Q2}\right\} \quad (15a)$$
$$\mathcal{U}_{CEC}=\left\{u_i|E_r^i> E_{r,Q2}\right\} \quad (15b)$$
$$\mathcal{U}=\mathcal{U}_{REC}\cup\mathcal{U}_{CEC} \quad (15c)$$

where E_r^i is the remaining energy of the EC u_i and:

$$E_{r,Q2}=\inf\left\{E_r^i, i=1,\dots,N|p\le F_{E_r^i}\right\} \quad (16)$$

is the quantile function, where $\inf\{\cdot\}$ is the *infimum* operator, p is equal to 2/3 for the upper 3-quantile index and $F_{E_r^i}$ represents the distribution of the remaining energy of all the ECs. Eq. (16) aims to select the remaining energy value that divides the set of ECs in 1/3 having a remaining energy higher than $E_{r,Q2}$ and 2/3 having a remaining energy lower than $E_{r,Q2}$. Note that we have considered one of the indexes (upper index) for the classification mainly due to two reasons. First it allows to classify the devices into two subsets. Moreover, the upper index allows to select higher number of users to act as the REC, due to the fact that each CEC can perform the computation for several RECs. RECs can only offload tasks to CEC or EN that are within their transmission range. Hence, EC u_i's set of CEC that are within its reach is given by

$$\mathcal{N}_{CEC}(i)=\left\{u_j|d(u_i,u_j)\le R_{\mathcal{U}},\forall u_j\in\mathcal{U}_{CEC}\right\}, \quad (17)$$

and the set of ENs that are within its reach is defined as

$$\mathcal{N}_{EN}(i) = \{f_m | d(u_i, f_m) \leq R_{\mathcal{F}}, \forall f_m \in \mathcal{F}\}$$

for all $u_i \in \mathcal{U}_{REC}$.

Besides, we denote the number of ECs in the REC subset, the number of CECs within reach of u_i, and the number of ENs within reach of u_i by $\bar{n} = |\mathcal{U}_{REC}|$, $\hat{n} = |\mathcal{N}_{CEC}(i)|$, and $\hat{f} = |\mathcal{N}_{EN}(i)|$, respectively.

4.1.2 Phase 2: Initial Solution Set Generation

The main idea of our initial solution generation algorithm is to form solutions that prioritize offloading tasks to ECs, or ENs, with low energy consumption and low task processing delay relative to other devices.

In particular, suppose a requesting EC $u_i \in \mathcal{U}_{REC}$ is seeking to offload one task with one portion. Then, to find an initial offloading decision for u_i, we first calculate the sum of energy consumption and task processing delay that u_i would experience by offloading its task portion to one of its neighboring CEC or ENs, i.e.,

$$\tilde{w}_{u_i}^{\nu_k^i} = \tilde{E}_{off}^{u_i,\nu_k^i} + \tilde{T}_{off}^{u_i,\nu_k^i} \tag{18}$$

for all $k \in [1, K_i]$, where $\tilde{E}_{off}^{i,\nu_k^i}$ and $\tilde{T}_{off}^{i,\nu_k^i}$ are obtained by normalizing E_{off}^{i,ν_k^i} and T_{off}^{i,ν_k^i} to the range $[0, 1]$ using min-max re-scaling [38], and K_i is the number of devices that can accept u_i's task portions. By further normalizing $\tilde{w}_{u_i}^{\nu_k^i}$ to be in the range $[0, \gamma_i]$, we define the upper bound on the number of the task portions for u_i that should be offloaded to device ν_k^i, by $w_{u_i}^{\nu_k^i} \in [0, \gamma_i]$.

Based on these upper bounds, the initial offloading decisions is as follows:

$$\phi_{u_i}^{\nu_k^i} \sim \text{unif}\{0, w_{u_i}^{\nu_k^i}\} \tag{19}$$

for all $k \in [1, K_i]$, and $u_i \in \mathcal{U}_{REC}$, where *unif* denotes the discrete uniform distribution between zero and $w_{u_i}^{\nu_k^i}$

However, since the offloading decisions in (19) may not satisfy constraint (12a), i.e., the sum of an EC's offloading decisions may not be equal to the total number of task portions $\sum_{k=1}^{K_i} \phi_{u_i}^{\nu_k^i} \neq \gamma_i$, we replace the offloading decisions in (19) as follows:

$$\phi_{u_i}^{max} \leftarrow \phi_{u_i}^{max} + \gamma_i - \sum_{k=1}^{K_i} \phi_{u_i}^{\nu_k^i} \tag{20}$$

where $\phi_{u_i}^{max} = \max_{\forall k \in [1,K_i]} \{\phi_{u_i}^{\nu_k^i}\}$.

The initial solution population $\mathbf{\Phi}_0 = \{\Phi_1, \Phi_2, ..., \Phi_Z\}$ is generated by repeating the above procedure for each offloading decision $\phi_{u_i}^{\nu_k^i} \in \Phi_z$ in every solution matrix Φ_z for all $\Phi_z \in \mathbf{\Phi}_0$. We summarize the proposed weighted initial solution generator in Algorithm 1.

Algorithm 1 Weighted initial solution generator

Input: $\tilde{E}_{off}^{u_i,\nu_k^i}, \tilde{T}_{off}^{u_i,\nu_k^i}$

Find $\tilde{w}_{u_i}^{\nu_k^i}$ using (18) for all $k \in [1, K_i]$ and $u_i \in \mathcal{U}_{REC}$
for $k = 1$ to K_i **do**
  Find $w_{u_i}^{\nu_k^i}$ by normalizing $\tilde{w}_{u_i}^{\nu_k^i}$ to the range $[0, \gamma_i]$, and rounding to the nearest integer.
  Set $\phi_{u_i}^{\nu_k^i}$ equal to a random integer drawn from the distribution unif$\{0, w_{u_i}^{\nu_k^i}\}$.
end for
Adjust the largest $\phi_{u_i}^{\nu_k^i}$ to meet constraint (12a) using (20)

Output: $\phi_{u_i}^{\nu_k^i}$

4.2 Selection

After generating the initial population, the proposed algorithm ranks the initial solutions based on their quality, and their distance to their nearest neighbors in the objective space. Specifically, the selection procedure first calculates the overall energy consumption and processing delay of solution $\Phi_z \in \mathbf{\Phi}$ as:

$$E_{\Phi_z} = \sum_{u_i \in \mathcal{U}_{REC}} \bar{E}_{off}^i \tag{21}$$

$$T_{\Phi_z} = \max_{u_i \in \mathcal{U}_{REC}} \{\bar{T}_{off}^i\} \tag{22}$$

for all $z \in [1, Z]$, where $\bar{E}_{off}^i$ and $\bar{T}_{off}^i$ are given by (5) and (10), respectively.

Then, using the constraint-dominance relationship explained in *Definition 2*, all individuals in $\mathbf{\Phi_0}$ are compared to each other and assigned a rank to determine the number of times each individual is dominated by the others. For example, an individual that is dominated by 10 other individuals is assigned a rank of 11, and a non-dominated individual receives a rank of 1.

An MOEA is desired to have a good diversity of non-dominated solutions converging to the Pareto-optimal front. To maintain this diversity, we further classify solutions based on the crowding distance which measures the proximity of an individual with others in the objective space, and denote it using I_z [9]. Crowding-distance calculation procedure is described in Algorithm 2.

Algorithm 2 Crowding-distance

Input: ϕ_z for $z \in [1, Z]$, C
for each ϕ_z, calculate the objective values $\omega_{z,1}, ..., \omega_{z,c}$ **do**
  Define individual I_z and set it to 0 for each individual ϕ_z
  for c=1 to C **do**
    Sort individuals ϕ_z in Z in ascending order according to $\omega_{z,c}$
    The crowding distance of the first and the last individual are set to infinity
    for z=2 to $Z-1$ **do**
      $I_z = I_z + \frac{\omega_{z-1,c} - \omega_{z+1,c}}{\max_z\{\omega_{z,c}\} - \min_z\{\omega_{z,c}\}}$
    end for
  end for
end for

Output: Crowding-distance of individual I_z

4.3 Reproduction

To generate a new set of possibly better solutions, the proposed algorithm performs the reproduction step through the following genetic operations: binary tournament, crossover, and mutation.

The algorithm first applies the binary tournament operation to the mating pool, which is the set of solutions that will be combined by the crossover operation to form new ones. We denote the mating pool by $\mathbf{\Phi}_0^M$. The binary operation consists in randomly selecting two solutions, Φ_i

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.2994232, IEEE Transactions on Mobile Computing

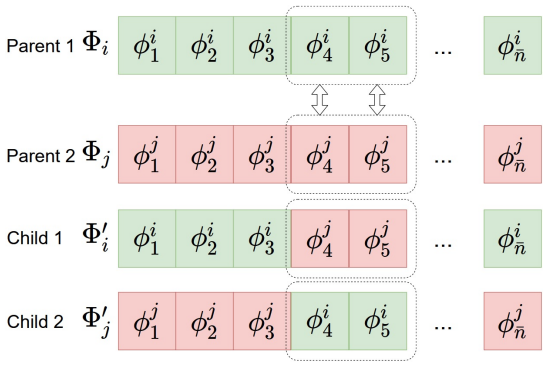


Fig. 3. Crossover operation on parent solutions.

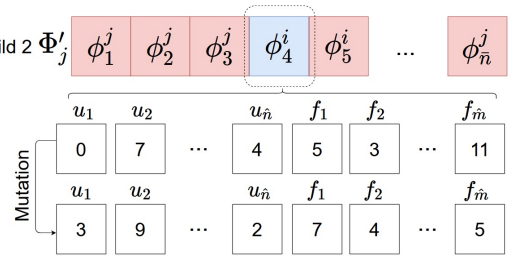


Fig. 4. Mutation operation on the selected solution (Child 2)

and Φ_j, from the current solution set $\mathbf{\Phi}_0$, and then adding the solution with the better dominance rank assigned during the selection phase to $\mathbf{\Phi}_0$. If both Φ_i and Φ_j have the same rank, the solution with larger crowding distance is selected. If both have equal crowding distance, one is chosen at random. The algorithm repeats the binary tournament until the maximum number m of solutions has been added to the mating pool.

Next, the crossover operation combines the solutions in the mating pool to generate new ones. The main idea of crossover is to randomly choose two solutions Φ_i and Φ_j from the mating pool $\mathbf{\Phi}_0^M$, and generate two child solutions Φ'_i and Φ'_j formed by combining parts of the parent solutions. Specifically, the crossover operation first randomly chooses the set of crossover points $\mathcal{D} = \{d_1, d_2, ..., d_D\}$ from the interval $[1, \bar{n}]$, where $d_i < d_{i+1}$ (for all $i \in [1, D]$). Then, children Φ'_i and Φ'_j are formed by alternating columns from Φ_i and Φ_j according to the crossover points, that is, $\Phi'_i = [\phi_1^i, ..., \phi_{d_1}^i, \phi_{d_1+1}^j, ..., \phi_{d_2}^j, ..., \phi_{d_{D-1}}^i, ..., \phi_D^i]$, and $\Phi'_j = [\phi_1^j, ..., \phi_{d_1}^j, \phi_{d_1+1}^i, ..., \phi_{d_2}^i, ..., \phi_{d_{D-1}}^j, ..., \phi_D^j]$, as shown in Fig. 3. The child solutions are added to the set of child solutions $\mathcal{Q}_0$. The crossover operation selects two solutions from the mating pool W times, where W is the maximum number of crossovers. However, the crossover operation is only performed with probability P_C.

To apply the mutation operation, we first randomly choose a child solution $\Phi'_i \in \mathcal{Q}_0$ with probability P_M. If a child solution is chosen, we replace a randomly chosen column with randomly generated offloading decisions. To ensure the new solution remains feasible, we apply (20). For example, in Fig. 4, the child solution Φ'_j has been chosen for mutation, and the offloading decisions in the 4th column have been replaced by random decisions, where $\gamma_4 = 30$.

4.4 Population Update

Once the offspring population $\mathcal{Q}_0$ has been generated and updated with mutations, we form the new solution population $\mathbf{\Phi}_1$ by discarding the low-quality solutions in the initial population $\mathbf{\Phi}_0$, and in the offspring population $\mathcal{Q}_0$. To this end, we first form an aggregate solution population $\mathcal{A}_0 = \mathbf{\Phi}_0 \cup \mathcal{Q}_0$, and calculate the rank and crowding distance of the solutions in $\mathcal{A}_0$ as described in Section 4.2. Then, we form the new population set $\mathbf{\Phi}_1$ by adding solutions from $\mathcal{A}_0$ in descending rank order until the maximum size of $\mathbf{\Phi}_1$ has been reached. In other words, we add solutions with rank 1 first, then, if there are unfilled positions in $\mathbf{\Phi}_1$, we add solutions with rank 2, and so on.

In the tth iteration of the algorithm, we apply the selection, and reproduction steps to the solution population $\mathbf{\Phi_t}$, generate the offspring set $\mathcal{Q}_t$, and the aggregate population $\mathcal{A}_t = \mathbf{\Phi}_t \cup \mathcal{Q}_t$. Then, the new solution population $\mathbf{\Phi}_{t+1}$ is filled with the top-ranked solutions in $\mathcal{A}_t$ as described above. The iteration continues until $t = X$. We summarize the proposed *NSGA2*-based algorithm in Algorithm 3.

Algorithm 3 Multi-objective offloading decision algorithm

Input: REC and CEC
Execute the initialization procedure
Generate the initial population $\mathbf{\Phi_0}$ of size Z, given the constraints (9)
for each $\phi_z \in \mathbf{\Phi_0}$ **do**
  for each $u_i \in \mathcal{U}_{REC}$ **do**
    calculate E_{ϕ_Z} using (18)
    calculate T_{ϕ_Z} using (19)
  end for
  $\mathcal{E} = \mathcal{E} + E_{\phi_Z}$
  $\mathcal{T} = \mathcal{T} + T_{\phi_Z}$
  Sorting $\mathbf{\Phi_0}$ based on Pareto dominance relationship
  Determining crowding distance to maintain the diversity
  Using binary tournament to fill the mating pool
  Apply crossover on $\mathbf{\Phi_0}$ and fill $\mathcal{Q}_0$
  Apply mutation on $\mathcal{Q}_0$
  Forming $\mathcal{A}_0 = \mathbf{\Phi_0} \cup \mathcal{Q}_0$
  for t = 1 to X **do**
    Form $\mathbf{\Phi_t}$ by adding sorted solutions from $\mathcal{A}_{t-1}$
    Generate $\mathcal{Q}_t$ by performing selection and reproduction on $\mathbf{\Phi_t}$
    Forming $\mathcal{A}_t = \mathbf{\Phi_t} \cup \mathcal{Q}_t$
  end for
end for
Output: Pareto Fronts to the MO computation sharing problem

5 Numerical Results

In this section, we present the numerical results obtained by our computer simulations. Specifically, we compare our *NSGA2*-based approach to other two benchmark algorithms. In the first benchmark, the algorithm assigns a task portion of equal size to each of the available devices (i.e., CECs and ENs). We call this straightforward solution the *fair allocation algorithm*. The second benchmark is a *distributed approach* already present in the literature which considers both delay and energy consumption in a similar scenario [37]. By comparing with the fair allocation algorithm and the *distributed approach*, we can show that our proposed NSGA2-based approach can reduce both the energy consumption and task completion delay of the ECs

TABLE 2
Simulation Parameters

Parameter	Value
Simulation Scenario Area	200m x 200m
Channel model	Outdoor RRH/Hotzone, Model 1: Pico to UE [40]
Channel Bandwidth	10 MHz
Max. number of solutions in mating pool (m)	100
Max. number of devices computing the offloaded task (K_i)	10
Crossover prob. (P_C)	0.8
Mutation prob. (P_M)	0.2
Max. number of crossovers (W)	100

TABLE 3
Task Parameters

Task Parameter	Value
Task size (D_s^i)	[1 5] MB
δ_{tx}, δ_{rx}	100 KB, 20 KB
Offloaded to downloaded portion (α)	5
Processing Application Operations	10 G FLOP per MB
Collecting Application Operations	1 G FLOP per MB

The computer simulations are performed in Matlab, where the considered parameters are listed in Table 2. The simulation is performed for 100,000 runs, where the duration of each run is set to 5 s, with the aim of comparing the performance in terms of average task delay, average EC energy consumption over time, and network lifetime, defined as:

- *Average Task Delay*: The average time spent for a task for transmitting, waiting, computing and receiving back the result over the number of generated tasks;
- *Average EC Energy Consumption Over Time*: The average energy all ECs have consumed per second;
- *Network Lifetime* $\kappa\%$: The time instant beyond which $\kappa\%$ of the ECs deplete their battery [39].

We hypothesize an area of 200×200 meters, with a variable number of heterogeneous ECs distributed randomly, while there are 5 ENs placed as seen in Fig. 1, in a way that each EC can be always served by at least one EN; ECs generate tasks according to a Bernoulli distribution with average $p = 0.1$ tasks per each simulation run. We have considered a battery capacity for all ECs equal to 5000 Joules; however, each EC has a starting random energy level uniformly distributed between 50% and 100% of the battery capacity. We have considered two applications: (i) a processing application generating tasks requiring a higher number of processing operations (e.g., image processing), and (ii) a collecting application, requiring a lower amount of processing operations, (e.g., sensor data analysis). In Table 3 the numerical values are reported, expressed in terms of Floating Point Operations (FLOP) per task data size. In order to have a realistic scenario, we have considered three ECs types with different capabilities, defined in Table 4.

In the *NSGA2*-based solution, the algorithm runs for 1000 iterations to find the Pareto results (i.e., $X = 1000$). In order to analyze the impact of the number of iterations on the energy and delay, we introduced a fourth approach in the simulation results, named *Constrained NSGA2*, with a lower computational complexity, where $X = 500$.

We first investigate the impact of the proposed initialization on the quality of the generated final solutions. To do so, we run the simulation for a single run for five different sets of ECs, and two cases: once when the *NSGA2*-based algorithm uses the Proposed Initialization (PI) approach, as introduced in Section 4.1, and once when it uses the Random Initialization (RI) approach. Then, we compare the average energy consumption and average processing delay values obtained from these two cases and show them in Table 5. As shown in Table 5, the PI approach shows better results in both energy consumption and task processing delay, when compared to the RI approach. Increasing the number of ECs in the network adds more complexity to the network and creates a bigger solution space, making it more time consuming and harder to find good quality solutions. However, the good quality initial solutions generated using the PI approach, helps the *NSGA2*-based algorithm to find better solutions even when the network complexity increases.

At each iteration of the simulation, the output of *NSGA2*-based approaches is composed by 15 non-dominated Pareto optimal solutions. To explore the different attributes of the Pareto optimal solutions, we considered three scenarios where once the best Pareto solution in terms of energy consumption is selected (Fig. 5), once the solution with the best delay is selected (Fig. 6), and once the trade-off solution with both relatively good delay and energy is selected (Fig. 7). Then, we analyze the average energy consumption (Figs. 5(a), 6(a), 7(a)), task delay (Figs. 5(b), 6(b), 7(b)), and the network life-time (Figs. 5(c), 6(c), 7(c)) for each of these three scenarios.

For the first scenario in which the selected Pareto solution has the best energy consumption (Fig. 5), we observe the lowest average energy consumption compared to the other two scenarios. Although as a result of giving preference to energy the delay attribute should be sacrificed (Fig. 5(b)), the *NSGA2*-based approaches still show better delay compared to the *Fair allocation* algorithm and *the distributed approach*. The same case is valid when we select the Pareto solution with the lowest delay (Fig. 6). In this case, the *NSGA2* approach shows the lowest average task delay comparing to the other approaches. However, we still observe that the *NSGA2* approach shows an acceptable average energy consumption, very close to the *Constrained NSGA2* approach. It must be noted that for these scenarios, the selected solution for the *Constrained NSGA2* is the trade-off point, where there is no preference between energy and delay. Finally, when the trade-off solution with relatively good delay and energy consumption is selected (Fig. 7), in both average task delay and energy consumption, *NSGA2*-based approaches perform better. However, the *NSGA2* approach performs better than the *Constrained NSGA2* approach, showing the impact of higher number of iterations that leads to higher variety of solutions generated when the number of iteration increases.

Furthermore, it must be noted that as the number of ECs increases, the number of interactions among ECs increases which is why the energy consumption figures tend to rise. In contrast to energy consumption, increasing the number of ECs will decrease the average task delay. Due to the high computational capability of the ECs and ENs for a single

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.2994232, IEEE Transactions on Mobile Computing

TABLE 4
Device Parameters

Parameter	Coverage Range	Battery Capacity	Initial Energy	Computational Capability	Computational Power	Idle Power	Transmission Power	Reception Power
High End EC	25 m	5000 J	[50 100]% battery capacity	25 G FLOPS	0.9 W	1.1 W	1.3 W	1.1 W
Low End EC	25 m	5000 J	[50 100]% battery capacity	15 G FLOPS	1.2 W	1.1 W	1.6 W	1.3 W
Heavy Duty EC	25 m	5000 J	[50 100]% battery capacity	20 G FLOPS	1.2 W	1.1 W	1.6 W	1.3 W
EN	100 m	-	-	150 G FLOPS	-	-	-	-

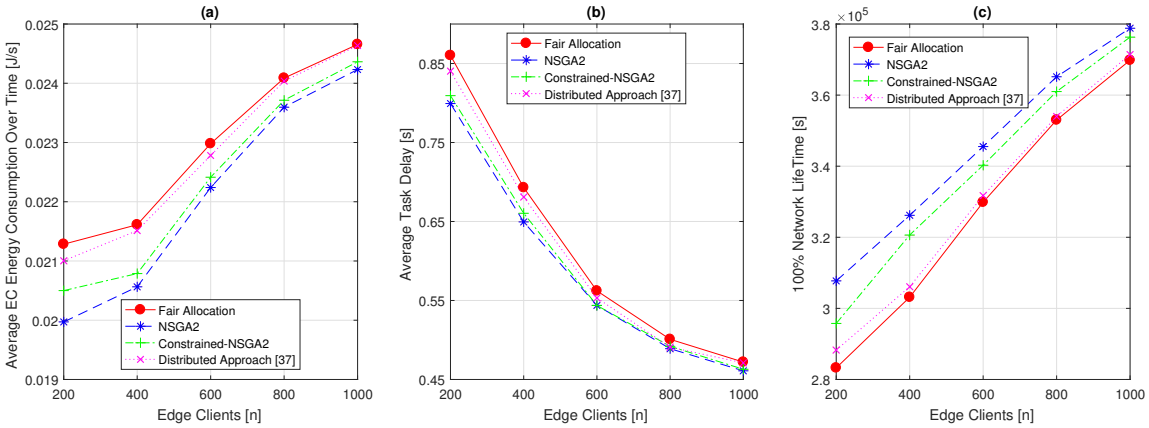


Fig. 5. Energy-based Pareto selection

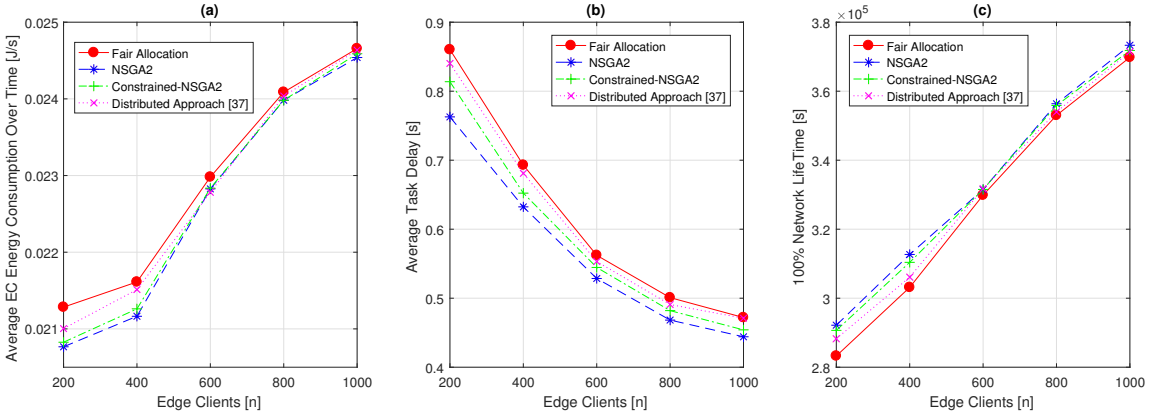


Fig. 6. Delay-based Pareto selection

TABLE 5
Impact of proposed initialization approach

Number of ECs	200	400	600	800	1000
Energy-PI (J/s)	0.0204	0.0211	0.0224	0.0237	0.0245
Energy-RI (J/s)	0.0209	0.0217	0.0232	0.0249	0.0260
Improvement (%)	6.1	5.3	3.5	3.2	2.6
Delay-PI (s)	0.7719	0.6510	0.5309	0.4711	0.4515
Delay-RI (s)	0.8244	0.6959	0.5702	0.5258	0.5098
Improvement (%)	12.9	11.6	7.4	6.9	6.8

task a small delay is obtained, and therefore the ratio of average task delay over number of generated tasks becomes smaller by the increase in number of ECs. To overcome the complexity of the algorithm due to higher number of ECs, we have used the energy-based ECs classification and weighted probabilistic solution generation, which helps the algorithm to generate approximately high-quality solutions faster with fewer number of solutions. Although the *constrained NSGA2* approach has half of the time complexity of the proposed *NSGA2* approach, it fails to deeply investigate the solution domain and this is why it shows comparably higher values for energy and delay.

Figs. 5c, 6c and 7c show the network lifetime. The figures are closely related with the energy consumption figures. As shown in the figures, exploiting NSGA2 for optimization results in less energy consumption, and therefore prolonging the network lifetime. Moreover, the best performance is gained when the Pareto solution is selected based on the lowest energy consumption.

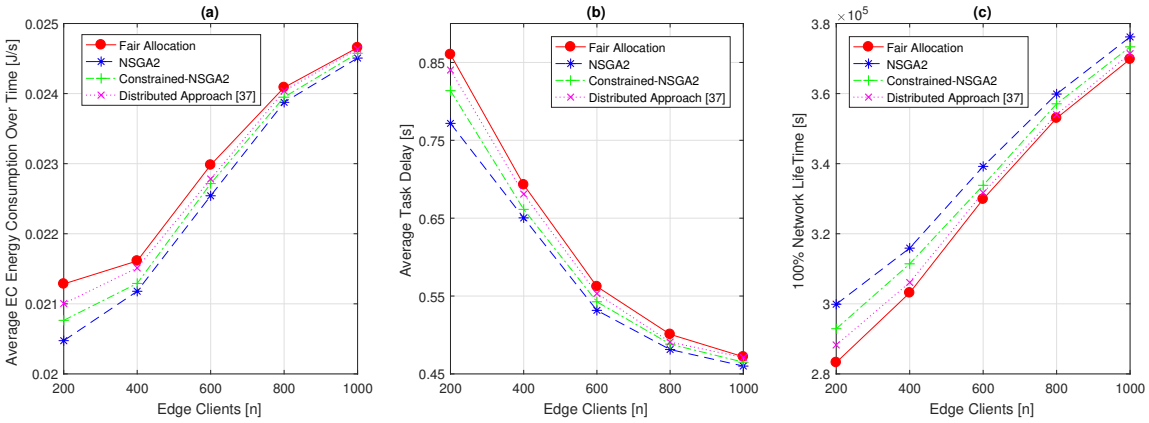


Fig. 7. Energy&Delay-based Pareto selection

It is worth mentioning that number of ECs and iterations have huge impact on the estimated portions to be offloaded to the available devices for computation. As seen in the previous figures, average task delay and average EC energy consumption are largely influenced by these two important factors. Hence, to show the trend of the changes in the solutions obtained by *NSGA2*-based algorithms, we have conducted the simulation for a single simulation run (i.e., 5 seconds) and analyzed the impact of the number of ECs and iterations on latency and energy consumption.

As seen in both Figs. 8a and 8b, as the number of ECs increases, the delay decreases and energy consumption increases. In particular, the higher number of task requests from RECs causes more interaction between the RECs and CECs, and subsequently, more energy consumption. In addition, the increase in the density of the RECs and CECs, leads to lower task processing delay. Furthermore, we can observe that increasing the number of iterations leads to improvement on energy consumption and processing delay of the tasks. This is due to the ability of the *NSGA2*-based to discover a variety of solutions, and generate possibly better ones using crossover and mutation operators.

However, the impact of number of iterations on the diversity of generated solutions degrades gradually and becomes less observable after some iterations. To analyze this impact, we fix the number of ECs to 1000 and plot the Pareto solutions of *NSGA2*-based for a single run, in Fig. 9. We can see from the figure that Pareto fronts for 200 iterations are more diverse and cover a higher range of values for energy consumption and task processing delay. As the number of iterations increases, this range becomes smaller, and reaches at its lowest for 1000 iterations.

We further analyze the generated Pareto fronts for different number of ECs and iterations, and show the results in Fig. 10. We can see the values of energy consumption and delay increase with the growth in number of ECs, while increasing the number of iterations helps slowing down this growth to some extent. We can see from the figure that increasing the number of iterations has more impact on the Pareto fronts, for the lower number of ECs in the network. Taking the case with 200 ECs as example, we see major improvement in energy consumption and task processing delay by increasing the number of iterations. With the growth in number of ECs, e.g., 1000 ECs, the solutions space will also grow significantly and makes it hard to find diverse solutions with better objective values. However, for the case with 1000 ECs we still observe that the *NSGA2*-based approach shows better energy consumption and task processing delay values, when compared to the *distributed approach* and the *fair allocation* approach.

6 Conclusion

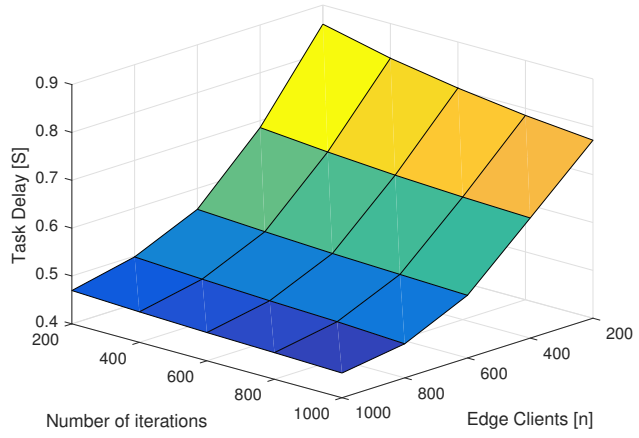
In this work, we have considered a computation sharing problem in MEC. The considered scenario is composed by ECs and ENs, where the ECs have the ability to offload their tasks to the nearby ECs and ENs. A computation sharing system model has been developed where the goal is to jointly minimize the energy consumption and the task delay when offloading the tasks portions to different devices. A suitable MOEA has been proposed for optimizing the computation sharing problem by considering both average task delay and average EC energy consumption. The numerical results shows that through the evolutionary steps of the NSGA2 it is possible to optimize both energy and delay and achieving better results than benchmark solutions. Moreover, the proposed NSGA2-based approach result in prolonging the network lifetime by optimizing the amount to be offloaded by the ECs. Furthermore, we have analyzed the effect of number of iterations in the genetic algorithm and have observed the convergence of the Pareto fronts in different scenarios with variable number of ECs. In the end, it can be noticed that the *Constrained-NSGA2* approach with a quite low time complexity can have results close to the *NSGA2* approach.

References

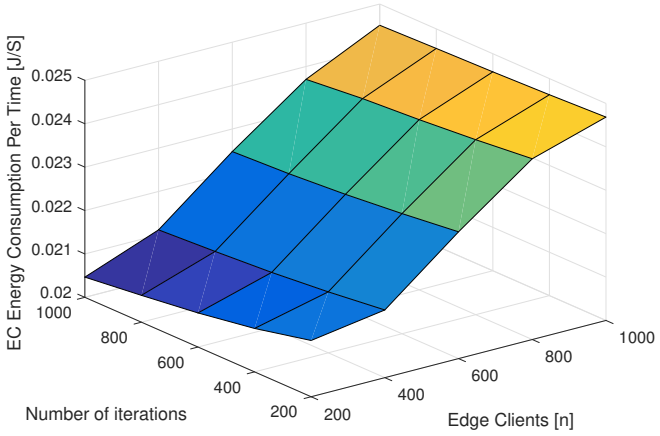
[1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 369–392, First Quarter 2014.

[2] A. u. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, First Quarter 2014.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, Aug. 2012, pp. 13–16.



(a) Latency behavior by different number of Edge Clients and Iterations



(b) Energy consumption behavior by different number of Edge Clients and Iterations

Fig. 8. Impact of Number of Iterations and number of Edge Clients on latency and energy consumption

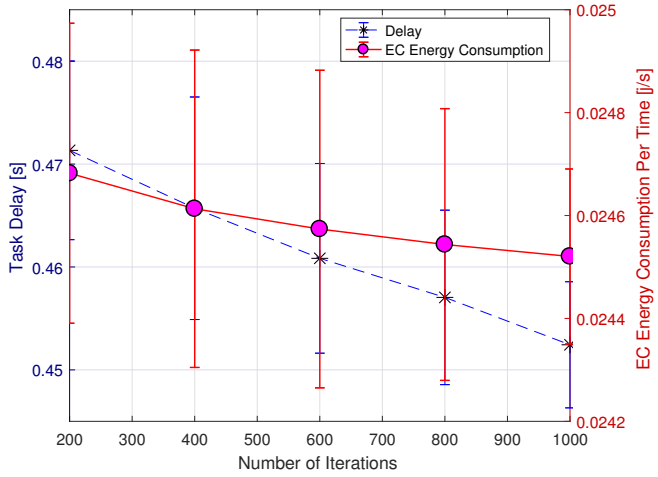


Fig. 9. Impact of Number of Iterations on latency and Energy Consumptions of the Pareto fronts

[4] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito, Eds. Singapore: Springer Singapore, 2018, pp. 103–130.

[5] M. Schneider, J. Rambach, and D. Stricker, "Augmented reality based on edge computing using the example of remote live support," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, Toronto, ON, Canada, Mar. 2017, pp. 1277–1282.

[6] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina, "MEC in 5G networks," ETSI, White Paper 28, Jun. 2018.

[7] Y. Cao, S. Chen, P. Hou, and D. Brown, "FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Boston, MA, USA, Aug. 2015, pp. 2–11.

[8] V. Stantchev, A. Barnawi, S. Ghulam, J. Schubert, and G. Tamm, "Smart items, fog and cloud computing as enablers of servitization in healthcare," *Sensors & Transducers*, vol. 185, no. 2, pp. 121–128, Feb. 2015.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[10] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Aug. 2014.

[11] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Toronto, ON, Canada, Apr. 2014, pp. 352–357.

[12] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[13] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 1451–1455.

[14] S. Chen, Y. Wang, and M. Pedram, "A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Atlanta, GA, USA, Dec. 2013, pp. 2885–2890.

[15] G. Li and J. Cai, "An online incentive mechanism for collaborative task offloading in mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 624–636, Jan 2020.

[16] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[17] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

[18] C. You, K. Huang, and H. Chae, "Energy efficient mobile cloud computing powered by wireless energy transfer," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1757–1771, May 2016.

[19] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 81–93, Jan. 2015.

[20] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. on Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr.-Jun. 2019.

[21] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for mobile edge computing," in *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, Shanghai, China, May 2018.

[22] J. Song, Y. Cui, M. Li, J. Qiu, and R. Buyya, "Energy-traffic tradeoff cooperative offloading for mobile cloud computing," in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, Hong Kong, China, May 2014, pp. 284–289.

[23] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, and P. Zhang, "Energy-aware

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2020.2994232, IEEE Transactions on Mobile Computing

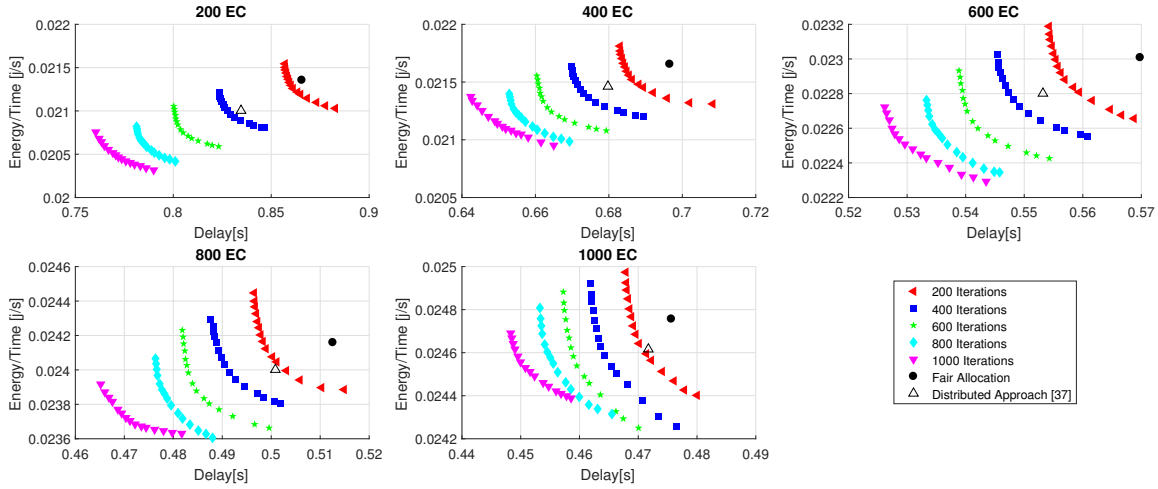


Fig. 10. Pareto fronts for different number of iterations and different number of Edge Clients.

mobile edge computation offloading for iot over heterogenous networks," *IEEE Access*, vol. 7, pp. 13 092–13 105, 2019.

[24] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, Nov. 2017.

[25] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[26] S. Hong and H. Kim, "QoE-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, London, UK, Jun. 2016.

[27] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *IEEE Access*, vol. 3, pp. 2306–2316, 2015.

[28] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug 2017.

[29] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Aug. 2017.

[30] J. Zhang, X. Hu, Z. Ning, E. C. . Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.

[31] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4188–4200, Jun. 2019.

[32] S. Chen, Y. Zheng, K. Wang, and W. Lu, "Delay guaranteed energy-efficient computation offloading for industrial IoT in fog computing," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, May 2019.

[33] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge QoE: Computation offloading with deep reinforcement learning for internet of things," *IEEE Internet Things J.*, mar 2020, early access.

[34] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.

[35] S. Midya, A. Roy, K. Majumder, and S. Phadikar, "Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach," *Journal of Network and Computer Applications*, vol. 103, pp. 58 – 84, Feb. 2018.

[36] L. Cui, C. Xu, S. Yang, J. Z. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, "Joint optimization of energy consumption and latency in mobile edge computing for internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791 – 4803, Jun. 2019.

[37] A. Bozorgchenani, D. Tarchi, and G. E. Corazza, "Centralized and distributed architectures for energy and delay efficient fog network based edge computing services," *IEEE Trans. Green Commun. Netw.*, vol. 3, no. 1, pp. 250 – 263, Mar. 2019.

[38] Y. Dodge, *The Oxford dictionary of statistical terms*. Oxford University Press on Demand, 2006.

[39] H. Yetgin, K. T. K. Cheung, M. El-Hajjar, and L. H. Hanzo, "A survey of network lifetime maximization techniques in wireless sensor networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 828–854, Second Quarter 2017.

[40] *Further advancements for E-UTRA physical layer aspects*, 3GPP TR 36.814, Rev. 9.0.0, Mar. 2010.

Arash Bozorgchenani was born in Rasht, Iran on 19th December 1989. He has received a B.Sc. degree in Information Technology in 2013, and a M.Sc degree in Information Technology with a major in Computer Networks in 2016 in Iran. Later he has received a Ph.D. degree in Electronics, Telecommunications and Information Technology in 2020 at University of Bologna, Italy, where currently he is a post-doctoral researcher. Arash has also been involved in the national research project Gaucho (PRIN 2015) in Italy. He has spent a 6-month research activity at university of Manitoba, Canada as a visiting PhD student. His research interests are in the areas of wireless communication, resource allocation and learning algorithms in wireless networks. He is working more specifically on fog/edge computing toward IoT and 5G applications. He is an IEEE student member since 2017.

Farshad Mashhadi received his BSc degree in Electrical engineering and Power Systems from the Azad University of Arak, Iran, in 2013, and the MSc degree in Computer Networks from the Azad University of Qazvin, Iran, in 2016. He is currently working toward his Ph.D. degree at cyber-physical system security lab (CPSSL) at the Wichita State University.

His research interests include cyber-physical systems, cloud computing, and optimization algorithms.

Daniele Tarchi [S'99, M'05, SM'12] was born in Florence, Italy, on 4th October 1975. He received his M.Sc. degree in Telecommunications Engineering and Ph.D. degree in Informatics and Telecommunications Engineering from the University of Florence, Florence, Italy, in 2000 and 2004, respectively.

From 2004 to 2010, he was a Research Associate with University of Florence, Italy. From 2010 to 2019 he was an Assistant professor at the University of Bologna, Bologna, Italy. He is now an Associate Professor at the University of Bologna, Italy. He is author of more than 100 papers, among which more than 30 on international journals. His research interests are mainly on resource allocation techniques, heterogeneous networks, Cognitive Radios and Networks, and Wireless Multimedia Networks, applied to both terrestrial and satellite wireless communications. Recently he has mainly focused on Smart City scenarios, multimedia systems, Fog Networks, and Smart Grid. He has been involved in several national and international research projects, and collaborates with several foreign research institutes.

Prof. Daniele Tarchi is an IEEE Senior Member since 2012. He is Editorial Board member for IEEE Wireless Communications Letters, IEEE Transactions on Vehicular Technology and IET Communications. He has been symposium co-chair for IEEE WCNC 2011, IEEE Globecom 2014, IEEE Globecom 2018 and IEEE ICC 2020, and workshop co-chair at IEEE ICC 2015.

Sergio Salinas received the BS degree in Telecommunications Engineering from Jackson State University, Jackson, in 2010, and the PhD degree in Electrical and Computer Engineering from Mississippi State University, Starkville, MS, in 2015. He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS. His research interests include security and privacy in cyberphysical systems, cloud computing, and big data. He is a member of the IEEE.