Al-Habob, Ahmed A.; Dobre, Octavia A.; Armada, Ana García; Muhaidat, Sami (2020). Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs. *IEEE Transactions on Vehicular Technology*, 69(8), pp.: 8805-8819.

DOI: https://doi.org/10.1109/TVT.2020.2995146

Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs

Ahmed A. Al-Habob, *Student Member, IEEE,* Octavia A. Dobre, *Fellow, IEEE,* Ana García Armada, *Senior Member, IEEE,* and Sami Muhaidat, *Senior Member, IEEE*

***Abstract*—In this paper, we consider parallel and sequential task offloading to multiple mobile edge computing servers. The task consists of a set of inter-dependent sub-tasks, which are scheduled to servers to minimize both offloading latency and failure probability. Two algorithms are proposed to solve the scheduling problem, which are based on genetic algorithm and conflict graph models, respectively. Simulation results show that these algorithms provide performance close to the optimal solution, which is obtained through exhaustive search. Furthermore, although parallel offloading uses orthogonal channels, results demonstrate that the sequential offloading yields a reduced offloading failure probability when compared to the parallel offloading. On the other hand, parallel offloading provides less latency. However, as the dependency among sub-tasks increases, the latency gap between parallel and sequential schemes decreases.**

***Index Terms*—Conflict graphs, genetic algorithms, mobile edge computing, parallel offloading, sequential offloading.**

I. Introduction

In recent years, there has been a tremendous growth of computationally-intensive mobile applications, such as 3D modeling, online gaming, and mobile augmented reality. However, mobile devices (such as tablets and smartphones) are normally constrained by the limited resources, e.g., computation capability of local central processing units (CPUs) and capacity-limited battery, and thus, restrict the users to fully enjoy highly computational demanding applications on their own devices. Mobile cloud computing (MCC) has emerged as a solution to provide elastic computing power to resource-constrained devices via offloading the computational-intensive tasks to powerful distant centralized servers [2], [3]. However, locating servers far away from the end-user has limitations, such as high transmission latency, communication bottlenecks, and security issues (e.g., some data should not be offloaded to servers that are located outside of national territory) [4]. Thus, MCC is not appropriate for mobile applications that are latency or security critical [5]. This motivated the idea of moving the function of clouds towards the network edges and gave birth to a new computing infrastructure, namely mobile-edge computing (MEC) [6], [7]. MEC provides lower offloading latency and jitter when compared to MCC. Moreover, while MCC is a centralized computation offloading approach, MEC provides a distributed approach which makes it a suitable solution for offloading computational-intensive and time-sensitive mobile applications [8], [9].

In MEC, based on local CPU availability or energy consumption consideration, mobile devices perform task computations locally or offload them to MEC servers. This decision plays a critical role in determining the computation efficiency, especially as the task offloading requires data transmission over the wireless channel. Two main computation task offloading policies can be found in the literature, namely partial offloading and binary offloading. In the former, a portion of the computation is performed locally at the mobile device and the other portion is offloaded to MEC servers [10]. In the latter, however, the mobile device determines whether a task should be computed locally or offloaded to MEC servers [11]. From the user's point of view, the most important requirements of task offloading are low energy consumption, low offloading error and low latency, mandating ultra-reliable and low-latency task offloading [12]. In recent communication networks, computing servers are deployed at the edge of the network to facilitate and accelerate connection. These servers are deployed in high numbers and have limited capabilities when compared to conventional, backbone cloud servers (which earned them the nickname of cloudlets) [13], [14].

Such dense deployment of the MEC servers gives the leverage of diversity and the ability of dividing a task into sub-tasks and offloading them to multiple MEC servers cooperatively to further shorten the latency [15]. In this respect, two main offloading schemes can be considered, namely sequential offloading and parallel offloading. For the former, the sub-tasks are offloaded in a time-sequential manner to servers over a shared communication channel [16], [17]. For the latter, the sub-tasks are offloaded simultaneously to servers over orthogonal communication channels [18] or using non-orthogonal multiple access [19]. Moreover, two important aspects should be considered in such offloading schemes: (1) The computational capabilities and channel qualities are different for the MEC servers. A low computation latency can be provided by an MEC server with high computational capability; however, it might encounter a high communication latency and high offloading failure probability due to a poor communication link between the mobile device and the MEC server; (2) Dividing the task into sub-tasks yields a

A.A. Al-Habob and O.A. Dobre are with the Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL, Canada (e-mail: {aaaalhabob, odobre}@mun.ca).

A.G. Armada is with the Department of Signal Theory and Communications, University Carlos III de Madrid, Leganés, Spain (e-mail: anagar@ing.uc3m.es.)

S. Muhaidat is with the Center for Cyber-Physical Systems, Department of Electrical and Computer Engineering, Khalifa University, Abu Dhabi, UAE (e-mail: muhaidat@ieee.org.)

This work has been accepted in part in the IEEE Globecom 2019 [1].

sophisticated scenario of dependency among sub-tasks. In many applications, the inter-sub-tasks dependency cannot be ignored, since it has a significant effect on the offloading and computation procedure. There are three main models of dependency among the sub-tasks, namely sequential dependency in which each sub-task depends on the output of the previous sub-task; parallel dependency in which a set of sub-tasks depends on the output of a previous sub-task; and general dependency in which a sub-task may depend on the output of one or more of the previous sub-tasks [20], [21]. Therefore, finding efficient scheduling decisions to assign a set of inter-dependent sub-tasks to a set of MEC servers is a challenging problem.

In the real world, the scheduling process is an approach through which a number of tasks are assigned to resources (servers) in order to complete the task execution process. Scheduling problems are typically formalized in terms of combinatorial optimization theory. Due to the high complexity and exponential growth search space of the scheduling problems, exhaustive search or random search methods are computationally demanding, thus rendering them impractical. In this context, the genetic algorithm (GA) was introduced as a robust global search method that searches for better solutions using a fitness score, which is obtained by evaluating an objective function without other derivative or auxiliary information [22], [23]. Other interesting scheduling methods, inherited from the graph theory, have emerged in the literature. These methods provide near optimum solutions with reduced computational complexity by avoiding the direct evaluation of the objective function in each iteration [24]–[26].

A. Related Work

Several research works (e.g., [18], [19], [27]–[32], and references therein) have proposed task offloading frameworks and algorithms to prolong the battery lifetime of the mobile devices. In [27], experimental results show that up to 50% of battery life can be preserved through remote processing of tasks. In [28] and [29], a single mobile device and single MEC server computation offloading problem was investigated. Wang *et al.* in [28] proposed partial computation offloading by jointly optimizing the consumed energy at the mobile device, offloading ratio, and computational delay. The authors in [29] investigated the optimal partial computation offloading jointly with the selection of constellation size and transmit power to optimize the consumed energy at mobile devices under latency constraint. A cooperative fog computing-based vehicular network architecture was proposed in [33], for the Internet-of-vehicles big data in a smart city. A dynamic network virtualization technique to achieve parallel computation in satellite-terrestrial networks was proposed in [34]. This technique integrates network resources and provides a cooperative parallel computation offloading model. In [35], a task scheduling approach with stochastic time cost for computation offloading was proposed to minimize the maximum tolerable delay by considering both the average delay and delay jitter. In [36], a multi-layer edge computing framework was proposed to assign tasks to each layer optimally. A heterogeneous multilayer MEC framework was proposed in [37], in which tasks that cannot be timely processed at the edge are offloaded to upper layer MEC servers and cloud center. The authors aimed at minimizing the offloading latency by jointly coordinating the task assignment, computing, and transmission resources in each layer.

The scenario of multiple mobile devices sharing a single MEC server for computation offloading was investigated in [18], [19], [30]–[32]. You *et al.* in [18] studied the total energy consumption minimization problem by considering both orthogonal frequency-division multiple access and time-division multiple access under latency constraint. An optimal priority policy is provided, which gives priority to mobile devices according to their local computing energy consumption and channel gains. In [19], a multiuser MEC system with one server was studied, in which users can simultaneously offload their computation tasks to a multi-antenna server over the same time/frequency resources based on non-orthogonal multiple access. The authors assumed both constant computational delay and downlink transmission delay, and focused their study on the energy consumption in the uplink phase. A joint optimization of computational and radio resources, aimed at optimizing mobile devices' energy consumption under power and latency constraints, was proposed in [30]. A distributed game theoretic approach for decision making to offload computation among multiple mobile devices was proposed in [31]. The authors showed that the game always admits a Nash equilibrium, achieves superior computation offloading performance, and scales well as the number of mobile devices increases. A cooperative offloading framework in which multiple mobile devices cooperate with each other to improve the computation capability of an MEC system was proposed in [32]. In all these works, a mobile device can be associated with one MEC server. However, in dense deployment of MEC servers, as envisioned in future networks, offloading a computational task to multiple nearby MEC servers can potentially improve the offloading process. In [15], a mobile device that offloads a set of independent tasks to a set of MEC servers in parallel via orthogonal sub-channels was studied. The authors aimed to minimize both mobile device's energy consumption and total tasks' execution latency. A sequential task offloading framework was proposed in [17]. In this framework, a mobile device segments a task into sub-tasks and offloads them to multiple servers in sequence. From a practical point of view, a task cannot be segmented arbitrarily and such a framework cannot handle the dependency among the sub-tasks.

In this paper, it is assumed that a delay-sensitive and computationally-intensive task is offloaded to multiple MEC servers. The motivation of this work is to jointly minimize the latency and offloading failure probability when offloading a set of inter-dependent sub-tasks to a set of MEC servers in both parallel and sequential offloading schemes. Accordingly, the main contributions of this paper are summarized as follows:

- Parallel and sequential offloading schemes of a delay-sensitive and computationally-intensive task to multiple

MEC servers are proposed. The task consists of a set of sub-tasks, and the general dependency among sub-tasks is considered.
- An optimization problem is formulated to jointly optimize the latency and offloading failure probability constrained over binary scheduling decision variables for both parallel and sequential offloading schemes.
- A solution based on GA is obtained for the formulated optimization problems.
- A computationally-efficient solution based on conflict graph models is additionally developed. In this solution, a greedy algorithm based on the minimum weighted vertex search algorithm is proposed to find the optimized offloading decisions for both parallel and sequential offloading schemes.

The remaining of this paper is organized as follows. Section II introduces the system model, along with the parallel and sequential offloading schemes. Section III illustrates an example to motivate the need for an optimized scheduling decision for both parallel and sequential task offloading schemes. In Section IV, the latency-reliability minimization problems are formulated for both parallel and sequential task offloading schemes. Section VI presents the proposed GA-based solution to find the optimized scheduling decision. A heuristic solution based on conflict graphs is introduced in Section VI. The time complexity of the proposed solutions is discussed in Section VII. Section VIII shows simulation results, and Section IX concludes the paper.

II. System Model and Offloading Schemes

In this section, the MEC system model is introduced first, followed by the latency and reliability analysis of the parallel and sequential offloading schemes. The notations used throughout this paper are listed in Table I.

A. System Model

A delay-sensitive and computationally-intensive task ($\mathcal{T}$) is offloaded by a mobile device to a set $\mathcal{S} = \{s_i\}_{i=1}^{I}$ of I MEC servers. Each server is equipped with a CPU that helps offloading and computing the task. A server s_i is represented by a tuple $s_i = \{Ru_i, Rd_i, f_i, p_i, q_i\}$, where Ru_i and Rd_i are the uplink and downlink data rates, respectively, f_i is the computational speed of the CPU in the i-th server (in cycles per second), and p_i and q_i are the packet error rate (PER) of the uplink and downlink, respectively. The size of the input data and output computed result of $\mathcal{T}$ are U and D packets, respectively, each packet of N_p bits. The task $\mathcal{T}$ consists of J $(J \leq I)$ sub-tasks $\mathcal{T} = \{\tau_j\}_{j=1}^{J}$. We use a tuple $\tau_j = \{u_j, c_j, d_j\}$ to represent the j-th sub-task in which u_j is the input data size (in packets), c_j is the number of CPU cycles that is required to process the sub-task, and d_j is the output computed result (in packets). The number of CPU cycles c_j is modeled as $c_j = \alpha_j N_p u_j$, where α_j (in cycles per bit) depends on the computational complexity of the sub-task. The uplink transmission delay and the computation delay of offloading the sub-task τ_j to the i-th server are $du_{ij} = \frac{u_j N_p}{Ru_i}$ and $dc_{ij} = \frac{c_j}{f_i}$, respectively. We define the offloading decision $\boldsymbol{\eta} = \left[\eta_{ij}\right]_{I\times J}$ such that:

$$\eta_{ij} = \begin{cases} 1, & \text{if } \tau_j \text{ is assigned to } s_i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Based on the offloading decision $\boldsymbol{\eta}$, the uplink transmission delay and the computation delay of the $j-$th sub-task τ_j can be expressed as $\mathcal{D}u_j(\boldsymbol{\eta}) = \sum_{i=1}^{I} \eta_{ij} du_{ij}$ and $\mathcal{D}c_j(\boldsymbol{\eta}) = \sum_{i=1}^{I} \eta_{ij} dc_{ij}$, respectively.

The dependency among the sub-tasks cannot be ignored in many applications, as it has significant effect on the offloading and computation procedure. We consider the general dependency model, in which the computation of a sub-task τ_l may depend on the output computed result of one or more of the previous sub-tasks [20], [21]. To address the inter-dependency among the sub-tasks, we introduce the sub-task dependency matrix $\mathbf{x} = \left[x_{lj}\right]_{J\times J}$ such that:

$$x_{lj} = \begin{cases} 1, & \text{if } \tau_j \text{ depends on the result of } \tau_l \, (l < j) \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The backhaul links among the servers are considered identical, reliable, and very high speed links. Accordingly, the transmission delay and failure probability of exchanging the intermediate results among servers is negligible.

B. Parallel Offloading Scheme

In the parallel offloading scheme, the mobile device offloads the sub-tasks to servers simultaneously via orthogonal sub-channels. The scheduling problem of this offloading scheme consists of assigning sub-tasks to servers under the following constraints:

- Each sub-task can be offloaded to only one server.
- Each server can handle only one sub-task.
- If $x_{lj} = 1$, then the sub-task τ_j can be offloaded to a server; however, the server holds the computing of τ_j until the computation of τ_l is finished.

A sub-task τ_j $(j > 1)$ may depend on one or more of the previous $(j-1)$ sub-tasks; as such, the server holds the computing of τ_j until the computation of all sub-tasks with $x_{lj} = 1$ in $\mathbf{x}$ is finished. The holding delay of the j-th sub-task can be expressed as:

$$\mathcal{D}hp_j(\boldsymbol{\eta}) = \max_{\forall l<j} \left\{ x_{lj} \left[\left(\mathcal{D}u_l(\boldsymbol{\eta}) + \mathcal{D}hp_l(\boldsymbol{\eta}) + \mathcal{D}c_l(\boldsymbol{\eta}) \right) - \mathcal{D}u_j(\boldsymbol{\eta}) \right]^{+} \right\}, \tag{3}$$

where $[v]^{+} \triangleq \max\{v, 0\}$. The downlink transmission delay of offloading the sub-task τ_j to the $i-$th server is $dd_{ij} = \frac{d_j N_p}{Rd_i}$. Based on the scheduler decision $\boldsymbol{\eta}$, the downlink transmission delay of the $j-$th sub-task τ_j can be expressed as $\mathcal{D}d_j(\boldsymbol{\eta}) = \sum_{i=1}^{I} \eta_{ij} dd_{ij}$. The delay of the parallel offloading and computing the j-th sub-task can be expressed as

$$\mathcal{D}p_j(\boldsymbol{\eta}) = \mathcal{D}u_j(\boldsymbol{\eta}) + \mathcal{D}c_j(\boldsymbol{\eta}) + \mathcal{D}hp_j(\boldsymbol{\eta}) + \mathcal{D}d_j(\boldsymbol{\eta}). \tag{4}$$

Notation	Description
I	Number of servers in the servers set $\mathcal{S}$
J	Number of sub-tasks
U	Input of the task $\mathcal{T}$ (packets)
D	Output of the task $\mathcal{T}$ (packets)
Rui (Rd_i)	Uplink (Downlink) data rate of server i (bits/s)
f_i	Computational speed of server i (in cycles per second)
p_i (q_i)	Uplink (Downlink) packet error rate of server i
u_j	Input data size of sub-task j (packets)
c_j	CPU cycles required to process sub-task j (cycles)
d_j	Output data size of sub-task j (packets)
α_j	Sub-task j computational complexity (cycles per bits)
N_p	Packet size (bits)
du_{ij} (dd_{ij})	Uplink (Downlink) delay of offloading τ_j to s_i
dc_{ij}	Computation delay of offloading τ_j to s_i
$\boldsymbol{\eta}$	Offloading decision
$\mathcal{D}u_j(\boldsymbol{\eta})$	Uplink transmission delay of sub-task j
$\mathcal{D}c_j(\boldsymbol{\eta})$	Computation delay of sub-task j
$\mathcal{D}d_j(\boldsymbol{\eta})$	Downlink transmission delay of sub-task j, parallel offloading
$\mathcal{D}_{D_k}$	Downlink transmission delay of the task, sequential offloading
$\mathbf{x}$	Sub-tasks dependency matrix
$\mathcal{D}hp_j(\boldsymbol{\eta})$ ($\mathcal{D}hs_j(\boldsymbol{\eta})$)	Holding time of sub-task j, parallel (sequential) offloading
$\mathcal{D}p_j(\boldsymbol{\eta})$	Offloading and computation delay of sub-task j
$\mathcal{L}p(\boldsymbol{\eta})$ ($\mathcal{L}s(\boldsymbol{\eta})$)	Latency of completing the task, parallel (sequential) offloading
$P_j^u(\boldsymbol{\eta})$ ($P_j^d(\boldsymbol{\eta})$)	Uplink (Downlink) failure probability of the sub-task j, parallel offloading
$P^u(\boldsymbol{\eta})$ ($P^d(\boldsymbol{\eta})$)	Uplink (Downlink) failure probability of the task, sequential offloading
$P_p(\boldsymbol{\eta})$ ($P_s(\boldsymbol{\eta})$)	Failure probability of offloading the task, parallel (sequential) offloading
$\Psi_p(\boldsymbol{\eta})$ ($\Psi_s(\boldsymbol{\eta})$)	Latency-reliability objective function, parallel (sequential) offloading
$\mathbf{g}_k$	Chromosome k
Ξ	Initial population size
M	Maximum number of offspring chromosomes
ζ_{p_k} (ζ_{s_k})	Chromosome $\mathbf{g}_k$ feasibility indicator, parallel (sequential) offloading
θ_{p_k} (θ_{s_k})	Chromosome $\mathbf{g}_k$ raw fitness, parallel (sequential) offloading
Θ_{p_k} (Θ_{s_k})	Chromosome $\mathbf{g}_k$ fitness value, parallel (sequential) offloading
v_{ij}	Vertex representing offloading of τ_j to s_i
$\mathcal{G}_p$ ($\mathcal{G}_s$)	Conflict graph, parallel (sequential) offloading
wp_{ij} (ws_{ij})	Vertex weight, parallel (sequential) offloading
Γ	Selected independent set

Table I: Main symbols used in the paper.

The total latency of completing the task with parallel offloading is given by

$$\mathcal{L}p(\boldsymbol{\eta}) = \max_{j\in\mathcal{T}}\{\mathcal{D}p_j(\boldsymbol{\eta})\}. \tag{5}$$

The uplink transmission failure probability and downlink transmission failure probability of offloading the j-th sub-task under the offloading decision $\boldsymbol{\eta}$ can be written as

$$P_j^u(\boldsymbol{\eta}) = 1 - \prod_{i=1}^{I}(1-p_i)^{\eta_{ij}u_j}, \tag{6}$$

and

$$P_j^d(\boldsymbol{\eta}) = 1 - \prod_{i=1}^{I}(1-q_i)^{\eta_{ij}d_j}, \tag{7}$$

respectively. The failure probability of offloading the j-th sub-task can be written as

$$P_j(\boldsymbol{\eta}) = P_j^u + \left[1 - P_j^u\right]P_j^d. \tag{8}$$

Consequently, the failure probability of offloading task $\mathcal{T}$ with the parallel offloading scheme can be expressed as

$$P_p(\boldsymbol{\eta}) = 1 - \prod_{j=1}^{J}\left(1 - P_j(\boldsymbol{\eta})\right). \tag{9}$$

An illustration example is provided in section III for ease of understanding.

C. Sequential Offloading Scheme

In the sequential offloading scheme, the mobile device offloads the sub-tasks to the servers in a time-sequential manner via a shared channel. The scheduling problem of this offloading scheme consists of assigning sub-tasks to servers under the following constraints:

- Each sub-task can be offloaded to only one server.
- Each server can handle (receive or compute) only one sub-task at any time instant.
- If $x_{lj} = 1$, then the sub-task τ_j can be offloaded to a server; however, the server holds the computing of τ_j until the computation of τ_l is finished.

In the sequential task offloading, the uplink transmission of the j-th sub-task will not start until uplink offloading of all previous $(j-1)$ sub-tasks is finished. In other words, the waiting time of the j-th sub-task before transmission is $\mathcal{W}_j(\boldsymbol{\eta}) = \sum_{l=1}^{j-1}\mathcal{D}u_l(\boldsymbol{\eta})$, $2 \le j \le J$, and $\mathcal{W}_1(\boldsymbol{\eta}) = 0$.[1]

[1]An illustration example is shown in Fig. 2.

A sub-task τ_j ($j > 1$) may depend on one or more of the previous $(j-1)$ sub-tasks. Since the server holds the computing of τ_j until the computation of all sub-tasks with $x_{lj} = 1$ in $\mathbf{x}$ is finished, the additional holding delay of the j-th sub-task with sequential offloading can be expressed as:

$$\mathcal{D}hs_j(\boldsymbol{\eta}) = \max_{\forall l<j} \left\{ x_{lj} \left[\mathcal{D}hs_l(\boldsymbol{\eta}) + \mathcal{D}c_l(\boldsymbol{\eta}) - \sum_{r=l+1}^{j} \mathcal{D}u_r(\boldsymbol{\eta}) \right]^+ \right\}. \quad (10)$$

The delay for offloading and computing the j-th sub-task can be expressed as

$$\mathcal{D}s_j(\boldsymbol{\eta}) = \mathcal{W}_j(\boldsymbol{\eta}) + \mathcal{D}u_j(\boldsymbol{\eta}) + \mathcal{D}c_j(\boldsymbol{\eta}) + \mathcal{D}hs_j(\boldsymbol{\eta}). \quad (11)$$

One server s_k is selected as a *sink server* to collect and send back the results to the mobile device. The total latency for completing the task is given by

$$\mathcal{L}s(\boldsymbol{\eta}) = \max_{j\in\mathcal{T}}\{\mathcal{D}s_j(\boldsymbol{\eta})\} + \mathcal{D}_{D_k}, \quad (12)$$

where $\mathcal{D}_{D_k} = \frac{DN_p}{Rd_k}$ is the downlink transmission delay of the resulted data D by the sink server s_k.
The failure probability of the uplink transmission of task $\mathcal{T}$ can be written as

$$P^u(\boldsymbol{\eta}) = 1 - \prod_{j=1}^{J} \left(1 - P_j^u(\boldsymbol{\eta})\right). \quad (13)$$

The failure probability of the downlink transmission of task $\mathcal{T}$ can be written as

$$P^d = 1 - (1-q_k)^D, \quad (14)$$

where the $k-$th server is selected as the sink server. Consequently, the task offloading failure probability with sequential offloading scheme can be expressed as

$$P_s(\boldsymbol{\eta}) = P^u(\boldsymbol{\eta}) + \left[1 - P^u(\boldsymbol{\eta})\right] P^d. \quad (15)$$

An illustration example is provided in the next section for ease of understanding.

III. Motivating Example

In this section, we discuss a numerical example to emphasize on the need for a good scheduling decision for both parallel and sequential task offloading schemes. Let us investigate a simple example in which the mobile device has a task $\mathcal{T}$, the input data and output computed result of $\mathcal{T}$ are $U = 1000$ and $D = 200$ packets, respectively. $\mathcal{T}$ consists of four sub-tasks, which can be represented as follows:
$\tau_1 = \{112, 0.85\times10^7, 23\}$; $\tau_2 = \{162, 2.48\times10^7, 33\}$;
$\tau_3 = \{589, 7.05\times10^7, 116\}$; $\tau_4 = \{137, 1.13\times10^7, 28\}$.
The sub-task dependency matrix associated with these sub-tasks is given as:

$$\mathbf{x} = \begin{array}{c} \begin{matrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{matrix}. \quad (16)$$

The server set consists of four servers $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, represented as follows:
$s_1 = \{4.3\times10^6, 5.2\times10^6, 7.1\times10^9, 0.16\times10^{-6}, 1.0\times10^{-7}\}$;
$s_2 = \{3.9\times10^6, 4.8\times10^6, 2.0\times10^9, 7.18\times10^{-6}, 1.2\times10^{-7}\}$;
$s_3 = \{5.5\times10^6, 6.5\times10^6, 3.6\times10^9, 0.10\times10^{-6}, 0.3\times10^{-7}\}$;
$s_4 = \{6.4\times10^6, 6.4\times10^6, 7.2\times10^9, 0.07\times10^{-6}, 0.4\times10^{-7}\}$.

- For the parallel offloading scheme, let us assume that the offloading decision $\boldsymbol{\eta}$ is selected such that:

$$\boldsymbol{\eta} = \begin{array}{c} \begin{matrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{matrix}. \quad (17)$$

  The total latency and offloading failure probability of the parallel offloading of the task with this offloading decision are 41.5 (ms) and 1.24×10^{-3}, respectively. This result leads the question of whether a better parallel offloading decision can be found to reduce both latency and offloading error. By examining all possible offloading decisions, the task can be offloaded in only 28.3 (ms) with an offloading failure probability of 8×10^{-4} with the following offloading decision:

$$\boldsymbol{\eta}_p^* = \begin{array}{c} \begin{matrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{matrix}. \quad (18)$$

  The offloading scheme of this preferable task offloading decision is shown in Fig. 1.

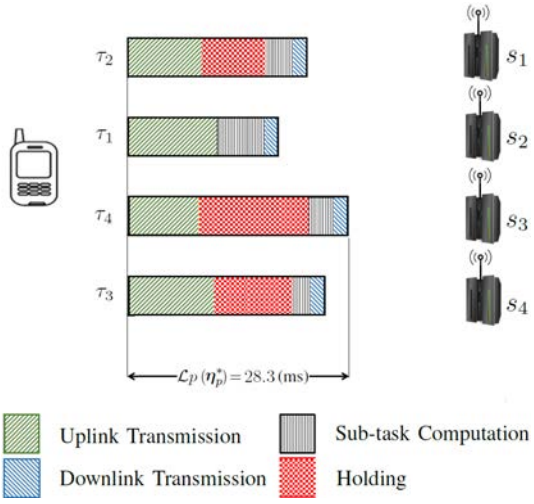


Figure 1: Parallel offloading scheme with offloading decision $\boldsymbol{\eta}_p^*$.

- For the sequential offloading scheme, let us assume that the offloading decision in (17) is adopted and s_4 is selected as the sink server. The total latency and offloading failure probability of the sequential offloading of the task with this offloading decision are 54.1 (ms)

and 1.25×10^{-3}, respectively. By examining all possible offloading decisions, the task can be offloaded in only 40.1 (ms) with the offloading failure probability of 8.2×10^{-5} with the following offloading decision:

$$\boldsymbol{\eta}_s^* = \begin{array}{c} \begin{matrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{matrix}, \tag{19}$$

and s_3 as the sink server. The offloading scheme of this preferable task offloading decision is shown in Fig. 2.

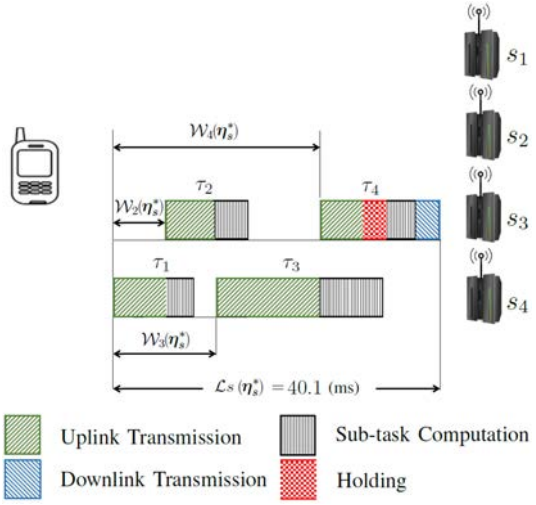


Figure 2: Sequential offloading scheme with offloading decision $\boldsymbol{\eta}_s^*$.

The question is how to find such preferable offloading decisions for both parallel and sequential offloading schemes in a systematic way for an arbitrary number of servers and sub-tasks. This motivates the problem formulations in the next section.

IV. Problem Formulations

A. Problem Formulation for Parallel Offloading Scheme

Our objective is to minimize both the total latency for completing a task and the offloading failure probability simultaneously. To tackle the trade-off between latency and reliability and to give them a similar significance, we consider the weighted product method [38], in which the latency-reliability cost function of the parallel offloading scheme can be formulated as[2]

$$\Psi_p(\boldsymbol{\eta}) = \mathcal{L}_p(\boldsymbol{\eta})\, P_p(\boldsymbol{\eta}). \tag{20}$$

Consequently, the optimization problem is formulated as

$$\mathbf{P1}\ \min_{\boldsymbol{\eta}}\ \Psi_p(\boldsymbol{\eta}), \tag{21a}$$

$$\text{s.t.}\quad \sum_{i=1}^{I}\eta_{ij} = 1,\ \forall j\in\mathcal{T}, \tag{21b}$$

$$\sum_{j=1}^{J}\eta_{ij} \le 1,\ \forall i\in\mathcal{S}, \tag{21c}$$

$$\eta_{ij}\in\{0,1\},\ \forall i\in\mathcal{S} \text{ and } j\in\mathcal{T}. \tag{21d}$$

Constraints (21b) and (21c) guarantee that each sub-task is offloaded to only one server, and no two sub-tasks are offloaded to the same server, respectively. In other words, at most one sub-task is offloaded to each server.

B. Problem Formulation for Sequential Offloading Scheme

The latency-reliability cost function of the sequential offloading scheme can be formulated as[3]

$$\Psi_s(\boldsymbol{\eta}) = \mathcal{L}_s(\boldsymbol{\eta}) P_s(\boldsymbol{\eta}). \tag{22}$$

The optimization problem is formulated as

$$\mathbf{P2}\ \min_{\boldsymbol{\eta}}\ \Psi_s(\boldsymbol{\eta}), \tag{23a}$$

$$\text{s.t.}\quad \sum_{i=1}^{I}\eta_{ij} = 1,\ \forall j\in\mathcal{T}, \tag{23b}$$

$$\eta_{i(j-1)} + \eta_{ij} \le 1,\ \forall i\in\mathcal{S},\ 2\le j\le J, \tag{23c}$$

$$\eta_{ij}\eta_{il}\left[\sum_{r=j+1}^{l-1}\mathcal{D}u_r(\boldsymbol{\eta}) - \left(\mathcal{D}c_j(\boldsymbol{\eta}) + \mathcal{D}hs_j(\boldsymbol{\eta})\right)\right] \ge 0, \tag{23d}$$

$$\forall i\in\mathcal{S},\ 1\le j\le J-2, j+2\le l\le J,$$

$$\eta_{ij}\in\{0,1\},\ \forall i\in\mathcal{S} \text{ and } j\in\mathcal{T}. \tag{23e}$$

Constraint (23b) guarantees that each sub-task is offloaded to only one server. Furthermore, constraints (23c) and (23d) guarantee that no two successive sub-tasks are offloaded to the same server and no sub-task is offloaded to a server that still handles another sub-task, respectively.
For the sequential offloading scheme, the sink server s_k can be selected such that

$$s_k = \arg\min_{i\in\mathcal{S}}\{\mathcal{D}_{D_i}\left(1-(1-q_i)^{D}\right)\}, \tag{24}$$

and for error free environment, $s_k = \arg\min_{i\in\mathcal{S}}\{\mathcal{D}_{D_i}\}$.

It is worth noting that the size of the search space for the optimization problems in **P1** and **P2** is 2^{IJ}. Consequently, the complexity of finding the optimum solution using exhaustive search is prohibitive for a reasonable number of servers and sub-tasks. The next two sections present more efficient methods to solve the problems using GA and conflict graph models, respectively.

[2]For error-free environment, the cost function is $\Psi_p(\boldsymbol{\eta}) = \mathcal{L}_p(\boldsymbol{\eta})$.

[3]For error-free environment, the cost function is $\Psi_s(\boldsymbol{\eta}) = \mathcal{L}_s(\boldsymbol{\eta})$.

V. Proposed Solution Using Genetic Algorithm

A genetic algorithm (GA) is a metaheuristic search technique, which evaluates a set of potential solutions called the initial population, each potential solution being referred to us chromosome. The GA generates new solutions through integrating the good features of existing solutions [39], [40].

In the proposed GA, a potential offloading decision is represented as a set of J parameters known as genes g_i, each gene representing an association of a sub-task and a server. These genes are joined together to form a string of integers known as a chromosome $\mathbf{g}$. A chromosome (J-dimensional vector of integer numbers) identifies the servers, as assigned to the vector elements represented by the sub-tasks. For example, the offloading decisions in (18) and (19) are represented by the following chromosomes

$$\mathbf{g}_p^* = \begin{matrix}\tau_1 & \tau_2 & \tau_3 & \tau_4\end{matrix} \\ [\,2 \quad 1 \quad 4 \quad 3\,], \tag{25}$$

and

$$\mathbf{g}_s^* = \begin{matrix}\tau_1 & \tau_2 & \tau_3 & \tau_4\end{matrix} \\ [\,4 \quad 3 \quad 4 \quad 3\,], \tag{26}$$

respectively.

A. Feasibility and Fitness for the Parallel Offloading Scheme

The chromosome representation described above ensures that constraints (21b) and (21d) are automatically satisfied since each sub-task is associated with exactly one server. However, this representation does not guarantee that constraint (21c) is fulfilled. A given chromosome is feasible for the parallel offloading scheme if its genes (servers) are distinct. In other words, if the number of the contributing servers is J. The feasibility indicator of a given chromosome for parallel offloading $\mathbf{g}_k$ is defined as

$$\zeta_{p_k} = \begin{cases} 1, & \text{if } \mid \bigcup_{i=1}^{J} g_i \mid = J, \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

It is clear that $\zeta_{p_k} = 0$ if and only if the chromosome $\mathbf{g}_k$ violates constraint (21c). We define the raw fitness of a chromosome $\mathbf{g}_k$ with parallel offloading scheme as

$$\theta_{p_k} = \frac{1}{\Psi_p\left(\boldsymbol{\eta}_k\right)}, \tag{28}$$

where $\boldsymbol{\eta}_k$ is the offloading decision associated with the chromosome $\mathbf{g}_k$. The fitness of a chromosome $\mathbf{g}_k$ is then defined as

$$\Theta_{p_k} = \zeta_{p_k}\theta_{p_k}. \tag{29}$$

Hence, the chromosome with the highest $\Theta_{p_k} > 0$ is feasible and minimizes the objective function in (21).

B. Feasibility and Fitness for the Sequential Offloading Scheme

The chromosome representation ensures that constraints (23b) and (23e) are automatically satisfied since each sub-task is associated with exactly one server. However, this representation does not guarantee that constraints (23c) and (23d) are fulfilled. Algorithm 1 returns the feasibility indicator for a given chromosome $\mathbf{g}_k$ in the sequential offloading scheme. In this algorithm, if $\xi' = 0$ then $\mathbf{g}_k$ violates (23c). A given chromosome is feasible for the sequential offloading scheme if Algorithm 1 returns $\zeta_{s_k} = 1$. We define the raw fitness of a chromosome $\mathbf{g}_k$ with sequential offloading scheme as

$$\theta_{s_k} = \frac{1}{\Psi_s\left(\boldsymbol{\eta}_k\right)}, \tag{30}$$

where $\boldsymbol{\eta}_k$ is the offloading decision associated with the chromosome $\mathbf{g}_k$. The fitness of a chromosome $\mathbf{g}_k$ is defined as

$$\Theta_{s_k} = \zeta_{s_k}\theta_{s_k}. \tag{31}$$

Hence, the chromosome with the highest $\Theta_{s_k} > 0$ is feasible and minimizes the objective function in (23).

Algorithm 1 Chromosome Feasibility for Sequential Offloading Scheme

input $\mathbf{g}_k$, du_{ij}, dc_{ij}, and $\mathbf{x}$
$\zeta_{s_k} = 1$
$\xi' = \prod_{j=1}^{J-1} \mid g_j - g_{j+1} \mid$
if $\xi' = 0$ **then**
  $\zeta_{s_k} \leftarrow 0$
  Return ζ_{s_k}
else
  Obtain $\boldsymbol{\eta}_k$ associates with $\mathbf{g}_k$
  $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta}_k$.
    if (23d) violated **then**
      $\zeta_{s_k} \leftarrow 0$
      Return ζ_{s_k}
    end if
end if
Return ζ_{s_k}

C. Genetic Algorithm

The adopted GA consists of the following steps:

1) Generate an initial population of Ξ randomly constructed chromosomes. Each chromosome in the initial population is generated by randomly associating a server with each sub-task.
2) Evaluate the fitness and the feasibility indicators of each chromosome. Two values are associated with each chromosome, namely the raw fitness value and the feasibility indicator [23].
3) Select two chromosomes as parents for reproduction. We adopt the binary tournament selection, in which two chromosomes are chosen from the population randomly and the one with the highest raw fitness value is selected

as a parent [23]. The same procedure is applied to select the other parent.

4) Generate a child chromosome from the parents by first applying a *crossover* operation. One-point crossover operation is adopted, in which a crossover point $1 \leq j \leq J$ is randomly selected. The child is structured from the first j genes which are taken from one of the parents, and from the remaining $J-j$ genes which are taken from the other parent. The crossover operation is followed by a *mutation* operation, in which two randomly selected genes in the child are exchanged (i.e., exchanging assigned servers between two randomly selected sub-tasks). After generating the child, the fitness and the feasibility indicator of the child chromosome are evaluated.
5) Replace a chromosome in the population by the child chromosome. The adopted replacement procedure is as follows. The chromosome with zero feasibility indicator and lowest raw fitness value is replaced by the child. If all the chromosomes in the population are feasible, the chromosome with the lowest raw fitness value is replaced by the child.
6) Repeat steps 3 and 4 until N offspring chromosomes have been generated without enhancing the best chromosome found so far or a maximum number of offspring chromosomes M has been generated.

It is worth mentioning that applying the GA described above to the motivating example in Section III leads to the offloading decisions in (18) and (19). However, it is known that GA suffers from the well-known curse of evaluating the fitness for each chromosome [40]. To avoid corresponding repeated expensive evaluations, next section introduces heuristic solutions based on conflict graph models.

VI. Proposed Solution using Conflict Graphs

In this section, we propose heuristic solutions to the formulated optimization problems using techniques inherited from graph theory. To find an optimized offloading decision, we should first design a representation of all feasible offloading schedules. We propose *offloading conflict graph* models that represent the offloading conflicts. The offloading conflict graph is an undirected graph in which each vertex has two indices representing an association of a server and a sub-task. Each undirected edge between two vertices is an offloading conflict between the two corresponding associations.

A. Conflict Graphs of the Parallel Offloading Scheme

The proposed conflict graph of the parallel offloading scheme $\mathcal{G}_p$ is constructed as follows:

Vertex Set: The vertex set consists of IJ vertices in which each vertex v_{ij} represents the offloading of sub-task τ_j to server s_i.

Scheduling Conflict Edges: Any two vertices v_{ij} and v_{kl} in $\mathcal{G}_p$ are set adjacent by a scheduling conflict edge if one of the following cases occurs:

1) $j = l \Rightarrow$ A sub-task cannot be handled by more than one server.
2) $i = k \Rightarrow$ A server cannot handle more than one sub-task.

Note that for the parallel offloading scheme, the holding time of a sub-task does not affect the feasibility of a solution. Given this configuration of the parallel offloading conflict graph, any independent set[4] of size J in $\mathcal{G}_p$ will represent a candidate feasible decision for the parallel offloading scheme. To select the offloading decision that provides minimum latency and guarantees minimum failure probability, we assign a weight wp_{ij} to each vertex in $\mathcal{G}_p$. This weight reflects both the latency and failure probability, being defined as

$$wp_{ij} = \left(du_{ij} + dc_{ij} + dd_{ij}\right)\left(1 - (1-p_i)^{u_j}(1-q_i)^{d_j}\right). \tag{32}$$

For error-free environment, the weight simply becomes

$$wp_{ij} = \left(du_{ij} + dc_{ij} + dd_{ij}\right). \tag{33}$$

B. Conflict Graphs of the Sequential Offloading Scheme

The proposed conflict graph of the sequential offloading scheme $\mathcal{G}_s$ is constructed as follows:

Vertex Set: The vertex set consists of IJ vertices in which each vertex v_{ij} represents the offloading of sub-task τ_j to server s_i.

Scheduling Conflict Edges: Any two vertices v_{ij} and v_{kl} in $\mathcal{G}_s$ are set adjacent by a scheduling conflict edge if one of the three cases below occurs:

1) $j = l \Rightarrow$ A sub-task or the feedback cannot be handled by more than one server.
2) $i = k$ and $\mid j - l \mid = 1. \Rightarrow$ Two successive sub-tasks cannot be offloaded to the same server.
3) $i = k$ and $l > j+1$ and $dc_{ij} + \widetilde{dh}_{ij} > \frac{\sum_{r=j+1}^{l-1} N_p u_r}{\max\limits_{\forall \varrho \in \mathcal{S}\setminus\{i\}} \{Ru_\varrho\}}$, where $\widetilde{dh}_{ij}$ is an approximation of the holding time of sub-task τ_l in server s_i, which can be expressed as

$$\widetilde{dh}_{ij} = \max_{\forall \sigma < j} \left\{ x_{\sigma j} \left[\widetilde{dh}_{i\sigma} + \mathcal{D}c_{i\sigma} - \left(\frac{\sum_{r=\sigma+1}^{j-1} N_p u_r}{\max\limits_{\forall \varrho \in \mathcal{S}\setminus\{i\}} \{Ru_\varrho\}} + \frac{N_p u_j}{Ru_i} \right) \right]^+ \right\} \tag{34}$$

$\Rightarrow$ A server cannot handle a new sub-task if it still computes another sub-task.

Given this configuration of the offloading conflicts, any independent set of size J in $\mathcal{G}_s$ will represent a candidate offloading decision. To select the offloading decision that provides minimum latency and guarantees minimum failure probability, we assign a weight ws_{ij} to each vertex in $\mathcal{G}_s$.

[4] An independent set in a graph is a set of vertices such as no edge exists between any pair of vertices in the set [41].

This weight reflects both the latency and failure probability, being defined as

$$ws_{ij} = \left(du_{ij} + dc_{ij}\right)\left(1 - (1 - p_i)^{u_j}\right). \quad (35)$$

For error-free environment, $ws_{ij} = \left(du_{ij} + dc_{ij}\right)$. The algorithm in the following section is designed to select the independent set that represents the offloading decision which provides minimum latency and guarantees minimum offloading failure probability.

C. Minimum Weighted Vertex Search Algorithm

In case of parallel or sequential offloading scheme, we perform the following substitutions $\mathcal{G} = \mathcal{G}_p$ and $w'_{ij} = wp_{ij}\ \forall v_{ij} \in \mathcal{G}_p$ or $\mathcal{G} = \mathcal{G}_s$ and $w'_{ij} = ws_{ij}\ \forall v_{ij} \in \mathcal{G}_s$, respectively. For a given sub-task τ_j, the vertex with the minimum weight $w'_{ij}\ \forall i \in \mathcal{S}$ represents the association between τ_j and the server that provides low latency and offloading failure error for offloading τ_j. Since each server has different channel state and computation speed and the association of an arbitrarily chosen sub-task with the best available server may prevent other highly demanding sub-tasks to be offloaded to the best available server, a given sub-task τ_j prioritizes to be associated with the best available server if:

- The sub-task τ_j is highly demanding (i.e., it consists of large number of packets and it requires a high number of CPU cycles). To sort the sub-tasks based on their demand, a raw prioritization weight is assigned to each sub-task such that $\Delta_j = w'_{ij}\ \forall j \in \mathcal{T}$ and i is fixed. Note that sorting the sub-tasks based on a weight calculated with respect to any $i \in \mathcal{S}$ leads to the same prioritized set of sub-tasks. This is because in each case, all sub-tasks will experience the same communication conditions and undergo the same computation capabilities. The sub-task with the highest $\Delta_j\ \forall j \in \mathcal{T}$ is the most demanding sub-task.
- A high number of sub-tasks depend on the output computed results of τ_j (i.e., τ_j has the largest $\kappa_j = \sum_{l=1}^{J} x_{jl}\Delta_l\ \ \forall j \in \mathcal{T}$).

Using this sub-task prioritization and the definition of the vertices weights, Algorithm 2 executes iteratively a greedy minimum weighted vertex search approach to select J-vertices independent set Γ, which represents the best offloading decision. We sort the sub-tasks in a descending order according to a prioritization weight of $\Delta_j + \kappa_j$. The index of the sorted sub-tasks is $j'(1), j'(2), j'(3), \ldots, j'(J)$ and the sub-task with $j'(1)$ has the higher priority to be associated with the best available server. Each iteration is implemented as follows: the vertex with the minimum weight in the vertices set $v^*_{ij'(l)}$ with $l = 1$ and $\forall\ 1 \leq i \leq I$ will be picked and added to Γ. The selected vertex $v^*_{ij'(l)}$ and all the vertices that are adjacent to it (symbolized in the algorithm by $\mathcal{N}_{\mathcal{G}}\left(v^*_{ij'(l)}\right)$) are eliminated from the graph $\mathcal{G}$. This elimination is performed to guarantee that the next picked vertex in the next iteration is not in scheduling conflict with the already selected ones in Γ. The algorithm continues by increasing l by one until $l = J$.

Algorithm 2 Minimum Weighted Vertex Search Algorithm.

1: **Input**: $\mathcal{G}_p$ and $wp_{ij}\ \forall v_{ij} \in \mathcal{G}_p$ OR $\mathcal{G}_s$ and $ws_{ij}\ \forall v_{ij} \in \mathcal{G}_s$. The sub-task dependency matrix $\mathbf{x}$.
2: Initialization:
- $\mathcal{G} \leftarrow \mathcal{G}_p$ OR $\mathcal{G}_s$
- $w'_{ij} \leftarrow wp_{ij}$ OR ws_{ij}
- Calculate $\Delta_j = w'_{1j}$ and $\kappa_j = \sum_{l=1}^{J} x_{jl}\Delta_l\ \forall j \in \mathcal{T}$
- Sort the sub-tasks in a descending order according to a prioritization weight of $\Delta_j + \kappa_j$
- The index of the sorted sub-tasks is $j'(1), j'(2), j'(3) \ldots, j'(J)$
- Set the selected independent set $\Gamma = \emptyset$

3: **for** $l = 1$ to J **do**
4: $v^*_{ij'(l)} \leftarrow \min_{i \in \mathcal{S}}\{w_{ij'(l)}\}$
5: $\Gamma = \Gamma \cup v^*_{ij'(l)}$
6: $\mathcal{G} \leftarrow \mathcal{G} \setminus \left(v^*_{ij'(l)} \cup \mathcal{N}_{\mathcal{G}}(v^*_{ij'(l)})\right)$
7: **end for**
8: **Return** Γ.

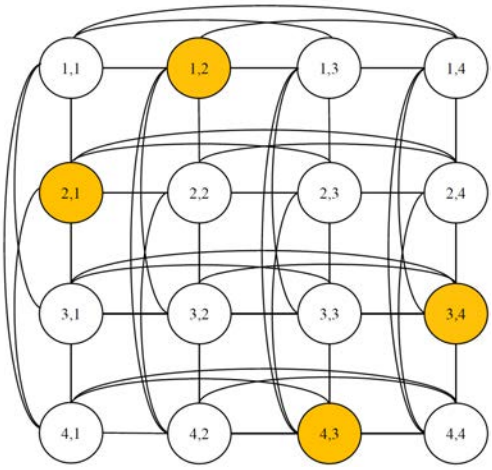


Figure 3: Offloading conflict graph of parallel offloading scheme of the example in Section III.

Figure 3 shows the offloading conflict graph of the parallel offloading scheme for the motivating example in Section III. For this graph, Algorithm 2 returns the independent set $\Gamma = \{v_{21}, v_{12}, v_{43}, v_{34}\}$ (represented by darker colour in Fig. 3). It is clear that the resulting offloading decision from this independent set is equivalent to that in (18).

Figure 4 shows the offloading conflict graph of the sequential offloading scheme of the example in Section III. For this graph, Algorithm 2 returns the independent set $\Gamma = \{v_{41}, v_{32}, v_{43}, v_{34}\}$ (represented by darker colour in Fig. 4). Obviously, the resulting offloading decision from this independent set is identical to that in (19).

VII. Complexity Analysis

This section introduces the computational complexity of the GA and the conflict-graph algorithms. The basic operations performed in the GA algorithm are selection operation, crossover operation, mutation operation, chromosome

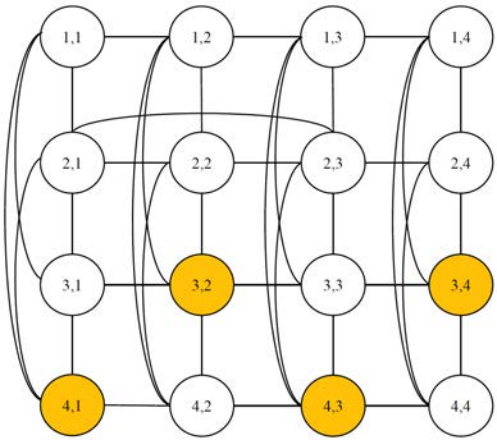


Figure 4: Offloading conflict graph of sequential offloading scheme of the example in Section III.

replacement, and fitness evaluation. The basic operations performed in the conflict-graph algorithm are generating the vertices set, building the adjacency edges, and finding the independent set.

A. Parallel Offloading

In the GA algorithm, generating a child chromosome requires the following operations. The crossover process has a complexity of $\mathcal{O}(J)$, the mutation process has also a complexity of $\mathcal{O}(J)$ [42], and chromosome replacement requires $\mathcal{O}(\Xi)$ operations. Consequently, generating the offspring chromosomes requires $\mathcal{O}\left([J+\Xi]M\right)$ operations. To evaluate the fitness of a chromosome, the following operations are required. Calculating the holding time of each sub-task requires $\mathcal{O}(J)$ operations. Consequently, calculating the total latency requires $\mathcal{O}\left(J^2\right)$ operations. Calculating the offloading failure probability requires $\mathcal{O}(IJ)$ operations. The time complexity of finding the feasibility indicator ζ_{p_k} is $\mathcal{O}(J)$. Since $J \leq I$, evaluating the fitness of a given chromosome has a time complexity of $\mathcal{O}\left(J^2+IJ+J\right) = \mathcal{O}(IJ)$. As we need to evaluate the fitness of each chromosome in the initial population and the offspring, the computational complexity of the GA with the parallel offloading scheme is $\mathcal{O}\left(IJ[\Xi+M]+[J+\Xi]M\right) = \mathcal{O}\left(IJ[\Xi+M]+\Xi M\right)$.

The vertex set size of the conflict-graph is $\mathcal{O}(IJ)$. To build the adjacency edges of the conflict-graph for the parallel offloading, all the vertices representing a given sub-task are connected to each other and all the vertices representing a given server are connected to each other. This means that we need a total of $\mathcal{O}(IJ+IJ) = \mathcal{O}(IJ)$ operations to build the adjacency edges. On the other hand, the complexity of the minimum weighted vertex search algorithm can be computed as follows. Computing the weights of the vertices requires $\mathcal{O}(IJ)$ operations. Sorting the sub-tasks has a time complexity of $\mathcal{O}(J\log J)$. The time complexity of selecting the minimum weighted vertex for each sub-task is $\mathcal{O}(I)$. Consequently, the complexity of the minimum weighted vertex search algorithm is $\mathcal{O}(IJ+J\log J+IJ) = \mathcal{O}(IJ)$ and the computational complexity of the conflict-graph algorithm is $\mathcal{O}(IJ+IJ) = \mathcal{O}(IJ)$. It is worth mentioning that representing the offloading decisions using (25)-(26) reduces the search space to $\mathcal{O}\left(I^J\right)$. Consequently, the computational complexity of finding the optimal solution using the exhaustive search is $\mathcal{O}\left(IJI^J\right) = \mathcal{O}\left(JI^{J+1}\right)$.

B. Sequential Offloading

Similar to the parallel offloading case, calculating the total latency and offloading failure probability for the sequential offloading scheme require $\mathcal{O}(J^2)$ and $\mathcal{O}(IJ)$ operations, respectively. The time complexity of finding the feasibility indicator ζ_{s_k} is $\mathcal{O}\left(IJ^2\right)$. Consequently, evaluating the fitness of a given chromosome has a time complexity of $\mathcal{O}\left(J^2+IJ+IJ^2\right) = \mathcal{O}\left(IJ^2\right)$ and the computational complexity of the GA with the parallel offloading scheme is $\mathcal{O}\left(IJ^2[\Xi+M]+[J+\Xi]M\right) = \mathcal{O}\left(IJ^2[\Xi+M]+\Xi M\right)$.

The vertex set size of the conflict-graph is $\mathcal{O}(IJ)$. To build the adjacency edges of the conflict-graph for sequential offloading, we set all the vertices representing a given sub-task connected to each other, which requires a total of $\mathcal{O}(IJ)$ operations. For all vertices representing a server we perform the following: (1) Connect any two vertices representing two successive sub-tasks, which requires $\mathcal{O}(J)$ operations; (2) Calculate $\widetilde{dh}_{ij}$ for each two vertices, which requires $\mathcal{O}(IJ)$ operations. This means that we need a total of $\mathcal{O}(IJ+[J+IJ]I) = \mathcal{O}(I^2J)$ operations to build the adjacency edges. Consequently, the computational complexity of the conflict-graph algorithm is $\mathcal{O}(I^2J+IJ) = \mathcal{O}(I^2J)$. It is worth mentioning that the computational complexity of finding the optimal solution using the exhaustive search is $\mathcal{O}\left(IJ^2I^J\right) = \mathcal{O}\left(J^2I^{J+1}\right)$.

Table II summarizes the time complexity of the algorithms. To introduce a quantitative measure of the complexity of the algorithms, Table II shows the average execution time of each algorithm on a personal computer equipped with an Intel(R) Core(TM) i5-4570 CPU, working at a clock frequency of 3.2 GHz, and 8 GB of RAM. The algorithms have been implemented in MATLAB. The system model consists of $I = 10$ servers, $J = 10$ sub-tasks, $\Xi = 10IJ$ chromosomes [43], and $N = 1000$ chromosomes, and $M = 10^5$ chromosomes. It is clear that the conflict graph solution is more complexity-efficient than that of the GA algorithm for both parallel and sequential offloading schemes.

Algorithm	Complexity	Av. Execution Time
GA-parallel offloading	$\mathcal{O}\left(IJ[\Xi+M]+\Xi M\right)$	4.4 ms
Graph-parallel offloading	$\mathcal{O}(IJ)$	1.3 ms
GA-sequential offloading	$\mathcal{O}\left(IJ^2[\Xi+M]+\Xi M\right)$	6.5 ms
Graph-sequential offloading	$\mathcal{O}(I^2J)$	1.8 ms

Table II: Algorithms complexity.

VIII. Simulation Results

In this section, we present simulation results to evaluate the performance of the proposed algorithms and to study the effect of the main parameters on the latency and offloading failure probability. In obtaining these results, the servers are uniformly distributed with distances from the mobile device

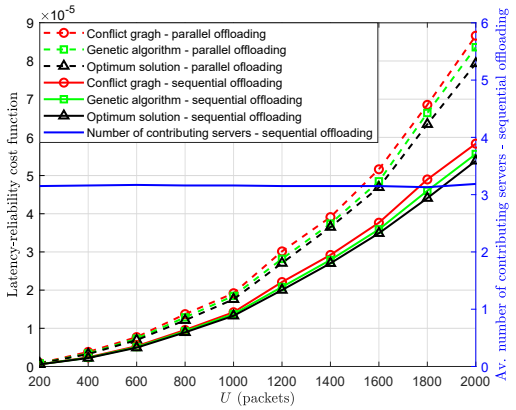
as 200 m $\sim$ 400 m. We set the channel gain as $h_i = \delta_i^{-\varepsilon}$, where δ_i is the distance between the mobile device and the i-th server and $\varepsilon = 4$ is the path loss exponent [31]. The channel bandwidth is 1 MHz and the spectral density of the additive noise is $N_0 = -173$ dBm/Hz. The transmit power of the mobile device and the servers is 100 mW [44]. For the parallel offloading scheme, orthogonal sub-channels are established each with 1 MHz bandwidth. The remaining parameters are as follows. The computation speed of the MEC server is generated randomly from a uniform distribution ranging from 2×10^9 to 8×10^9 cycles per second [15], the packet size is $N_p = 20$ bytes [45], task input and output data are $U = 1000$ packets [17], [46] and $D = 200$ packets [44], respectively, and the computational complexity of a sub-task α_j in cycles per bit follows a Gamma distribution with shape and scale parameter of 4 and 200, respectively [47], [48]. To represent the dependency among the sub-tasks, a given sub-task depends on any of the previous sub-tasks with probability π. The initial population size is $10IJ$ chromosomes [43], and N and M equal 10^3 and 10^5, respectively. The default simulation parameters, which are summarized in Table III, are considered in the results, unless otherwise stated.

Parameter	Value
Channel bandwidth	1 MHz
Noise spectral density N_0	-173 dBm/Hz
Server transmit power	100 mW
Mobile device transmit power	100 mW
Packet size	20 bytes
Computing speed of the servers	$[2;\ 8] \times 10^9$ cycles/s
Total task input data size U	1000 packets
Total task output data size D	200 packets
Computational complexity parameter	$\alpha_j \sim$ Gamma $(4, 200)$ cycles/bit
Sub-tasks dependency probability π	0.3
Initial population size Ξ	$10IJ$ chromosomes
N	10^3 chromosomes
M	10^5 chromosomes

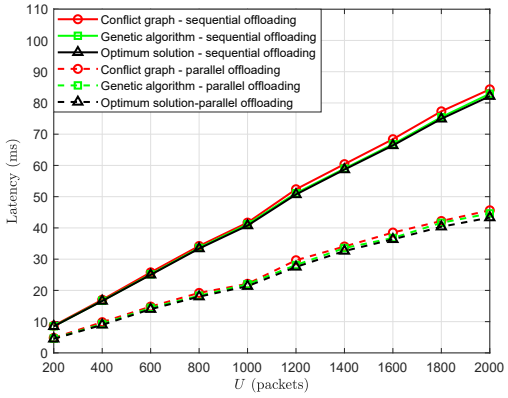
Table III: Simulation parameters.

Figure 5a illustrates the performance of the GA, conflict graph, and optimum solution (obtained using exhaustive search) versus the task data size U. It is seen that the proposed algorithms achieve near-optimum performance, and the latency-reliability cost function of the sequential offloading is less than that of the parallel offloading scheme. To gain deep insight into this result, Figs. 5b and 5c show the corresponding latency and offloading failure probability, respectively. It is clear that the parallel offloading provides less latency; however, sequential offloading provides less offloading failure probability. For parallel offloading, all servers contribute to offloading simultaneously, which leads to less latency; however, not all of them have good channel quality, which yields higher failure probability. On the other hand, in sequential offloading, the scheduler explores the best available servers and less servers are contributing.

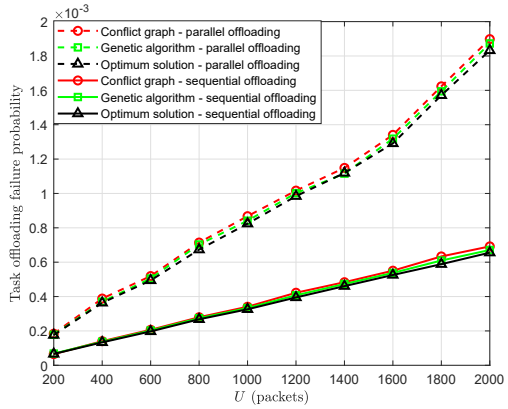
Figure 6a illustrates the effect of the inter-sub-tasks dependency on the latency-reliability cost function. It is noticed that the cost function and number of contributing servers in the sequential offloading are not affected considerably by the inter-sub-tasks dependency. Also, it is clear that as the probability of inter-sub-tasks dependency increases, the



(a) Latency-reliability cost function versus the task data size U.



(b) Latency versus the task data size U.

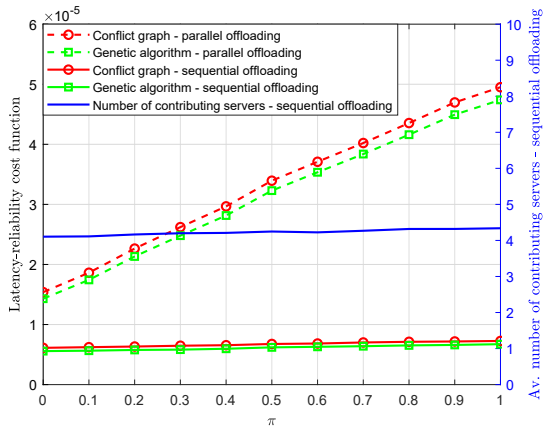


(c) Offloading failure probability versus the task data size U.

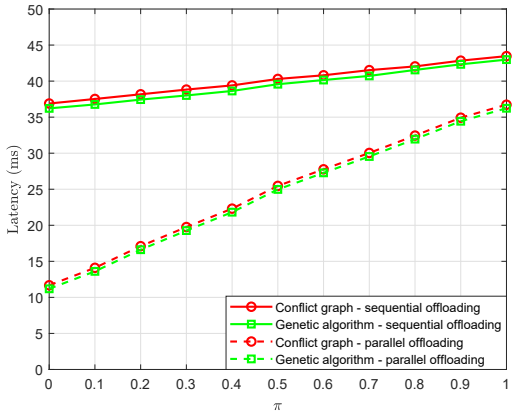
Figure 5: The effect of the task data size U with $I = 6$ servers and $J = 6$ sub-tasks.

cost function of the parallel offloading increases. To gain better insight into this behavior, Figs. 6b and 6c portray the corresponding latency and offloading failure probability, respectively. It is clear that there is no noticeable change in the offloading failure probability. However, as the probability of inter-sub-tasks dependency increases, the latency of the parallel offloading increases rapidly and the latency response gap between sequential and parallel schemes decreases. The reason is that while all sub-tasks are offloaded simultaneously in the parallel scheme, the sub-task that depends on another

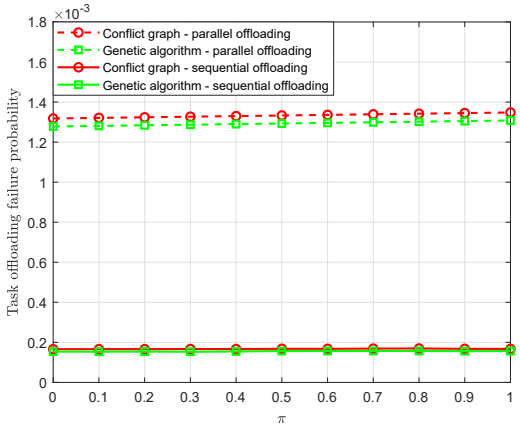
sub-task will be held, and the simultaneous offloading loses its main advantage. On the other hand, in sequential offloading, the sub-task waits until all the previous sub-tasks are offloaded and their computing has already begun. Consequently, the waiting time due to the inter-sub-tasks dependency is less in comparison with the parallel offloading.



(a) Latency-reliability cost function versus the inter-sub-task dependency probability π.



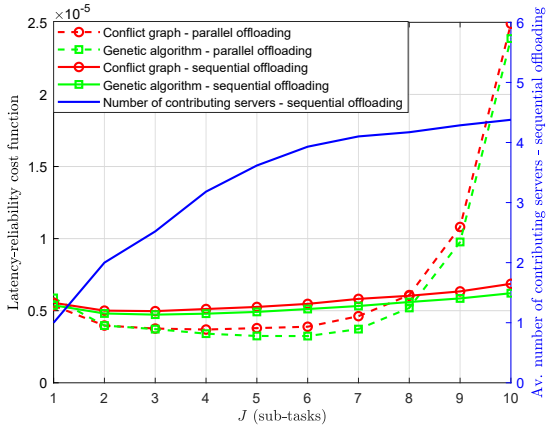
(b) Latency versus the inter-sub-task dependency probability π.



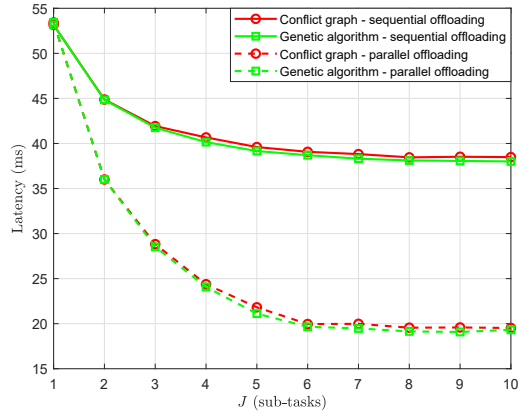
(c) Offloading failure probability versus the inter-sub-task dependency probability π.

Figure 6: The effect of the inter-sub-task dependency probability π with $I = 10$ servers and $J = 10$ sub-tasks.

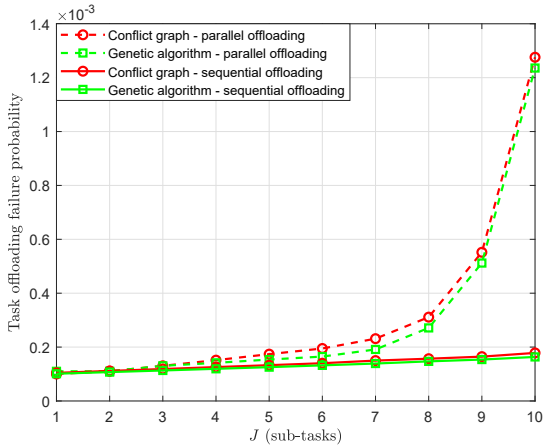
Figure 7 shows the effect of the number of sub-tasks J with a total of $I = 10$ available servers. It is noticed that as the number of sub-tasks increases, the latency reduces for both parallel and sequential offloading schemes. The offloading failure probability increases remarkably for the parallel offloading. This can be attributed to the fact that as the number of sub-tasks increases, each sub-task is offloaded to a server and not all servers have the same quality of connections.



(a) Latency-reliability cost function versus the number of sub-tasks.



(b) Latency versus the number of sub-tasks.



(c) Offloading failure probability versus the number of sub-tasks.

Figure 7: The effect of the number of sub-tasks J with $I = 10$ servers.

In Fig. 8, all servers are located at the same distance from the mobile device (i.e., $\delta_i = \delta \ \forall s_i \in \mathcal{S}$). Such scenario is suitable for the case where servers are located in close proximity to each other or they are placed at the arc of a circle whose center is the mobile device. It is noticed that as the distance between the servers and the mobile device increases (i.e., the SNR decreases), the latency increases in both parallel and sequential offloading schemes. However, for the sequential offloading, the latency grows more rapidly. This can be attributed to the fact that reducing the SNR will increase the uplink transmission delay, and consequently, the waiting time of the sub-tasks[5] also increases. It is also noticed that as the distance between the servers and the mobile device increases (i.e., uplink transmission delay increases), the number of contributed servers in sequential offloading decreases. This is because during the long time duration of uplink transmission of a sub-task, more servers will complete the computation of their sub-task and get ready to contribute. Such availability of servers gives the scheduler the ability to offload more sub-tasks to less number of servers (because it selects the best available servers).

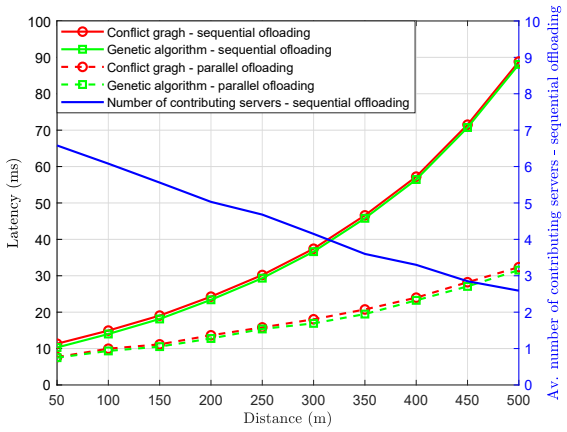


Figure 8: Latency versus distance with $I = 10$ servers and $J = 10$ sub-tasks.

IX. Conclusion

We proposed task scheduling for parallel and sequential offloading of a delay-sensitive and computationally-intensive task to multiple MEC servers. The problem was formulated as the joint optimization of latency and offloading failure probability. Heuristic solutions based on genetic algorithm and conflict graph models were developed. Simulation results revealed that the proposed solutions provide performance close to the optimal one, with reduced complexity. Also, findings showed that sequential offloading provides less offloading failure probability and requires a lower number of servers. On the other hand, parallel offloading provides less latency. However, as the dependency among sub-tasks increases, the latency response gap between sequential and parallel schemes decreases.

[5]The waiting time of a sub-task is an accumulation of the uplink delay of all previous sub-tasks, i.e., $\mathcal{W}_j(\boldsymbol{\eta}) = \sum_{l=1}^{j-1} \mathcal{D}u_l(\boldsymbol{\eta})$.

References

[1] A. A. Al-habob, O. A. Dobre, and A. G. Armada, "Sequential task scheduling for mobile edge computing using genetic algorithm," in *Proc. IEEE Globecom, Waikoloa, HI, USA*, 2019, pp. 1–6.

[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] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts*, vol. 22, no. 1, pp. 38–67, Mar. 2020.

[4] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, pp. 1–40, Jun. 2018.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth quarter 2017.

[6] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, Nov. 2014.

[7] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Third quarter 2017.

[9] A. A. Al-habob and O. A. Dobre, "Mobile edge computing and artificial intelligence: A mutually-beneficial relationship," *IEEE ComSoc Techncial Committees Newsletter*, Nov. 2019.

[10] J. Ren, G. Yu, Y. Cai, Y. He, and F. Qu, "Partial offloading for latency minimization in mobile-edge computing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[11] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[12] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. John Wiley & Sons, 2012.

[13] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[14] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sep. 2018.

[15] 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.

[16] A. A. Al-Habob, A. Ibrahim, O. A. Dobre, and A. G. Armada, "Collision-free sequential task offloading for mobile edge computing," *IEEE Commun. Lett.*, vol. 24, no. 1, pp. 71–75, Jan. 2020.

[17] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12 825–12 837, Feb. 2018.

[18] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[19] F. Wang, J. Xu, and Z. Ding, "Multi-antenna NOMA for computation offloading in multiuser mobile edge computing systems," *IEEE Trans. Commun.*, pp. 1–14, Nov. 2018.

[20] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 352–357.

[21] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput., Early Access*.

[22] P. Vasant, *Meta-heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance*. Information Science Reference, 2013.

[23] P. C. Chu and J. E. Beasley, "A genetic algorithm for the generalised assignment problem," *Comput. Oper. Res.*, vol. 24, no. 1, pp. 17–23, Apr. 1997.

[24] A. A. Al-Habob, Y. N. Shnaiwer, S. Sorour, N. Aboutorab, and P. Sadeghi, "Multi-client file download time reduction from cloud/fog storage servers," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1924–1937, Aug. 2018.

[25] A. Douik, H. Dahrouj, T. Y. Al-Naffouri, and M. Alouini, "Distributed hybrid scheduling in multi-cloud networks using conflict graphs," *IEEE Trans. Commun.*, vol. 66, no. 1, pp. 209–224, Jan 2018.

[26] A. A. Al-Habob, S. Sorour, N. Aboutorab, and P. Sadeghi, "Conflict free network coding for distributed storage networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2015, pp. 5517–5522.

[27] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "The remote processing framework for portable computer power saving," in *Proc. ACM Symp. Appl. Comp., San Antonio, TX*, 1999.

[28] 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.

[29] P. Di Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile edge computing," *arXiv preprint arXiv:1307.3835*, 2013.

[30] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[31] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[32] Z. Sheng, C. Mahapatra, V. C. M. Leung, M. Chen, and P. K. Sahu, "Energy efficient cooperative computing in mobile wireless sensor networks," *IEEE Trans. on Cloud Comput.*, vol. 6, no. 1, pp. 114–126, Jan. 2018.

[33] W. Zhang, Z. Zhang, and H. Chao, "Cooperative fog computing for dealing with big data in the Internet of vehicles: Architecture and hierarchical resource management," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 60–67, Dec. 2017.

[34] Z. Zhang, W. Zhang, and F. Tseng, "Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques," *IEEE Network*, vol. 33, no. 1, pp. 70–76, Jan. 2019.

[35] W. Zhang, Z. Zhang, S. Zeadally, and H. Chao, "Efficient task scheduling with stochastic delay cost in mobile edge computing," *IEEE Commun. Lett.*, vol. 23, no. 1, pp. 4–7, Jan. 2019.

[36] C. Yao, X. Wang, Z. Zheng, G. Sun, and L. Song, "EdgeFlow: Open-source multi-layer data flow processing in edge computing for 5G and beyond," *IEEE Network*, vol. 33, no. 2, pp. 166–173, Mar. 2019.

[37] P. Wang, Z. Zheng, B. Di, and L. Song, "HetMEC: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4942–4956, Aug. 2019.

[38] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, Apr. 2004.

[39] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, Oct. 1988.

[40] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, 2000, vol. 7.

[41] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*, Chapman and Hall/CRC, 2013.

[42] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (GA) based load balancing strategy for cloud computing," *Proc. Technol.*, vol. 10, pp. 340–347, Dec. 2013.

[43] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. IEEE North American Fuzzy Information Processing*, 1996, pp. 519–523.

[44] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.

[45] C. She, C. Yang, and T. Q. S. Quek, "Cross-layer optimization for ultra-reliable and low-latency radio access networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 1, pp. 127–141, Jan. 2018.

[46] E. Bastug, M. Bennis, M. Medard, and M. Debbah, "Toward interconnected virtual reality: Opportunities, challenges, and enablers," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 110–117, Jun. 2017.

[47] 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.

[48] 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.