Future Generation Computer Systems 135 (2022) 205–222

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks

Huned Materwala [a,b], Leila Ismail [a,b,*], Raed M. Shubair [c], Rajkumar Buyya [d]

[a] Intelligent Distributed Computing and Systems (INDUCE) Research Laboratory, Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Al Ain, Abu Dhabi 15551, United Arab Emirates
[b] National Water and Energy Center, United Arab Emirates University, Al Ain, Abu Dhabi 15551, United Arab Emirates
[c] Department of Electrical and Computer Engineering, New York University (NYU), Abu Dhabi, United Arab Emirates
[d] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 29 September 2021
Received in revised form 6 April 2022
Accepted 9 April 2022
Available online 21 April 2022

Keywords:
Computation offloading
Edge–cloud computing
Energy-efficiency
Evolutionary genetic optimization algorithm
Quality of service (QoS)
Vehicular Ad Hoc Networks (VANET)

ABSTRACT

Vehicular Ad Hoc Networks (VANET) is an emerging technology that enables a comfortable, safe, and efficient travel experience by providing mechanisms to execute applications related to traffic congestions, road accidents, autonomous driving, and entertainment. The mobile vehicles in VANET are characterized by low computational and storage capabilities. In such scenarios, to meet applications' performance requirements, requests from vehicles are offloaded to edge and cloud servers. The high energy consumption of these servers increases operating costs and threatens the environment. Energy-aware offloading strategies have been introduced to tackle this problem. Existing works on computation offloading focus on optimizing the energy consumption of either the IoT devices/mobile/vehicles and/or the edge servers. This paper proposes a novel offloading algorithm that optimizes the energy of edge–cloud integrated computing platforms based on Evolutionary Genetic Algorithm (EGA) while maintaining applications' Service Level Agreement (SLA). The proposed algorithm employs an adaptive penalty function to incorporate the optimization constraints within EGA. Comparative analysis and numerical experiments are carried out between the proposed algorithm, random and genetic algorithm-based offloading, and no offloading baseline approaches. On average, the results show that the proposed algorithm saves 2.97 times and 1.37 times more energy than the random and no offloading algorithms respectively. Our algorithm has 0.3% of violations versus 52.8% and 62.8% by the random and no offloading approaches respectively. While the energy-non-SLA-aware genetic algorithm saves, on average, 1.22 times more energy than our approach, however, it violates SLAs by 159 times more than our proposed approach.

1. Introduction

Vehicular Ad Hoc Networks (VANET) [1] is an emerging technology where vehicles acting as network nodes are equipped with computational resources and connectivity such as vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-roadside (V2R), vehicle-to-sensors (V2S), vehicle-to-pedestrian (V2P), and vehicle-to-everything (V2X) communications. It enables a comfortable, safer, convenient, and efficient travel experience for users by using applications such as sending alerts for congestions and accidents, autonomous driving, video-enabled real-time navigation, interactive gaming, and entertainment [2–4]. These applications often require high computation and storage resources, and low latency to process complex operations. However, mobile vehicles have limited onboard computing and storage capabilities to process resource-intensive applications while maintaining the Quality of Services (QoS). To address this issue, a cloud-based vehicular network has been introduced. Cloud computing [5,6] provides on-demand computational and storage resources to mobile vehicles over the Internet. The remote cloud servers have high computation capabilities that would satisfy applications processing times. However, a high latency between the vehicle and cloud resources hinders the deployment of time-critical applications such as autonomous driving. In addition, a delayed response for applications such as traffic congestion and interactive gaming becomes less reliable. To overcome vehicles-to-cloud latency issues, Vehicular Edge Computing (VEC) [7] has been introduced.

* Corresponding author at: Intelligent Distributed Computing and Systems (INDUCE) Research Laboratory, Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Al Ain, Abu Dhabi 15551, United Arab Emirates.
E-mail address: leila@uaeu.ac.ae (L. Ismail).

https://doi.org/10.1016/j.future.2022.04.009

VEC provides the computing resources to the vehicles at the edge of the radio access networks to support applications with low-latency requirements [8]. VEC servers deployed within the Roadside Units (RSU) improve applications' QoS in terms of throughput and latency. However, edge servers have less computational and storage resources compared to cloud resources, often making edge servers bottlenecks with increasing demands from vehicles. Consequently, it becomes crucial to take advantage of heterogeneous resource capabilities available at vehicles, edge, and the cloud layers to meet the requirements of the applications. Therefore, computational offloading mechanisms can help to either execute applications at the vehicles/edge layers or remote cloud resources [9]. However, offloading strategies should consider the high energy consumption issue of the edge–cloud integrated computing platform [10–12], and applications' requirements [13–16]. It is estimated that data centers consisting of thousands of computing servers will consume 4.5% of the total energy consumption globally by 2025 [17]. Moreover, it is predicted that the Information and Communications Technology (ICT) industry will account for 14% of the global carbon emissions by 2040 [18]. Consequently, it becomes crucial to address the issue of high energy consumption of edge–cloud layers in vehicular networks while offloading.

Several works in the literature have proposed energy-aware computation offloading for an IoT-edge–cloud integrated computing system [19–30]. However, these works focus on optimizing the energy consumption of either the IoT devices/vehicles and/or the edge servers. To the best of our knowledge, no work in the literature has focused on optimizing the energy consumption of the edge and cloud servers simultaneously while maintaining applications' Service Level Agreements (SLA). Our previous results on energy savings in edge and cloud systems using deterministic approaches are promising [10–12,31]. However, these approaches become computationally expensive while offloading a set of applications' requests. This is because offloading a set of applications' requests to edge and cloud servers is a Nondeterministic Polynomial-time (NP) hard problem where the search space and time to optimal solution increase exponentially with increasing requests and servers [20]. Empirical evaluations have shown the effectiveness of the evolutionary algorithm in finding a solution for the NP-hard scheduling problem in a cloud computing environment [32,33]. In this paper, a computation offloading algorithm using Evolutionary Genetic Algorithm (EGA) is proposed that optimizes the energy consumption of edge–cloud servers, while maintaining applications SLAs in terms of latency and processing time. The proposed algorithm executes a vehicle's request locally on the edge server to which the request has been submitted or offloads the request to one of the cloud servers. The offloading decision is made in a way that the total energy consumption of all the requests is minimized while maintaining each request's SLA in terms of latency and processing time requirements. The main contributions of this paper are summarized as follows:

- A novel algorithm is proposed to solve the problem of offloading in vehicular networks that minimizes the total energy consumption of vehicles' requests while adhering to latency and processing time constraints.
- This optimization NP-hard problem is solved by applying an adaptive penalty function to our evolutionary-based algorithm, to obtain a near-optimal solution in polynomial time.
- The performance of the proposed algorithm is evaluated and compared with three algorithms in terms of total energy consumption and the percentage of SLA violations. The algorithms are evaluated using different constraints requirements and vehicular network characteristics.

The rest of the paper is organized as follows. Section 2 presents an overview of related work. Our edge–cloud integrated computing system model for vehicular networks is described in Section 3. The formulation of the offloading optimization problem is outlined in Section 4. Section 5 presents the proposed EGA-based energy-SLA-aware offloading algorithm. Numerical experiments and comparative performance results with baseline methods are provided in Section 6. Finally, Section 7 concludes the paper with future research directions.

2. Related work

Several works in the literature have proposed energy-aware computation offloading for an IoT-edge–cloud integrated computing system [19–30]. Table 1 presents a comparison between these works and shows the system component(s) on which a request from an IoT device/mobile/vehicle is processed, the considered component(s) for energy optimization, the constraints on SLA requirements, and whether or not the time for the delivery of request's response is included while computing the request's total execution time. As shown in the table, most of the works focus on optimizing the energy consumption of the IoT device/mobile/vehicle [19,20,24–30]. On the other hand, few works [21,22] focus on optimizing the energy consumption of the edge servers, while only one work [23] considers optimizing the energy of both mobile devices and edge servers. No work focuses on optimizing the energy consumption of the edge–cloud integrated computing system. In this paper, an EGA-based algorithm is proposed to offload the vehicles' requests either to edge or cloud servers in a way that the total energy consumption of the requests is minimized while maintaining the requests' SLA requirements in terms of latency and processing time.

As shown in Table 1, Li et al. [19] proposed an iterative search-based energy-aware offloading algorithm to execute requests on either IoT devices or an edge server. The algorithm aims to minimize the energy consumed by IoT devices while transmitting the requests to the edge server with the SLA constraint on the request's total execution time, i.e., summation of the request's transmission and processing times. Similarly, Guo et al. [20] proposed an iterative search-based offloading algorithm to execute requests either locally on IoT mobile devices or offload them to an edge server. This is to minimize the weighted sum of the request's total execution time (both computation and communication) and mobile device's energy consumption while maintaining the deadline constraint. A similar approach to minimize the request's execution time and the energy consumption of mobile devices, with an SLA constraint on execution time, using an iterative search method is proposed by Zhang et al. [24]. The algorithm enables mobile devices to execute requests either locally or offload them to an edge server. The execution time involves processing and communication times. However, an iterative search-based algorithm [19,20,24] generates identical offloading solutions in a subset of solutions leading to premature convergence [34].

In [25], Li et al. proposed deep reinforcement learning (DRL)-based offloading algorithm to execute requests either locally on a mobile device or offload them to an edge server such that the total energy consumption of the mobile device is the minimum. The algorithm is constrained by the request's total execution time, i.e., processing and communication. With a similar objective, Huang et al. [26] proposed a DRL-based offloading algorithm to execute a request either locally on a vehicle or an edge server such that the vehicle's energy consumption (processing and communication) is the minimum while maintaining the request's deadline constraint. However, the computational complexity of DRL [25,26] increases with increasing edge/cloud servers and users' requests [35].

Table 1
Related work on energy-aware computation offloading for IoT-edge–cloud integrated computing system.

Work	Algorithm	Considered component(s) for request processing			Considered component(s) for energy optimization			Considered SLA requirements			Request–response delivery time consideration
		IoT device/mobile/vehicle	Edge server	Cloud server	IoT device/mobile/vehicle	Edge server	Cloud server	Latency	Processing time	Total execution time (deadline)	
[19]	Iterative search-based	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗
[20]		✓	✓	✗	✓	✗	✗	✗	✗	✓	✗
[24]		✓	✓	✗	✓	✗	✗	✗	✗	✓	✗
[25]	Deep reinforcement learning	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗
[26]		✓	✓	✗	✓	✗	✗	✗	✗	✓	✓
[27]	Lyapunov optimization	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
[28]		✓	✓	✗	✓	✗	✗	✗	✗	✗	✓
[29]	Memetic algorithm	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
[30]	Non-dominated sorting genetic algorithm-II	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗
[21]	–	✗	✓	✗	✗	✓	✗	✗	✗	✓	✗
[22]	Non-dominated sorting genetic algorithm-II	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗
[23]	Heuristic approach	✓	✓	✗	✓	✓	✗	✗	✗	✓	✗
This paper	Evolutionary genetic algorithm	✗	✓	✓	✗	✓	✓	✓	✓	✗	✓

Another energy-aware offloading algorithm is proposed by Huang et al. [27] using Lyapunov optimization that allows vehicles to execute requests locally or to offload them to an edge server such that the vehicles' energy consumption and packet drop rate are the minimum. Pu et al. [28] used Lyapunov optimization-based algorithm to execute a request either locally on a vehicle, by a group of communicating vehicles, or by an edge server such that the vehicle's energy consumption is the minimum while maintaining an SLA constraint on the request's deadline. The vehicle's energy consumption in [27,28] involves request processing and transmission. However, the Lyapunov-based approaches [27,28] are best suited for problems involving optimization of long-term performance where each performance metric in the objective function must be time-averaged, for instance, minimizing average energy consumption. Consequently, they are not suitable for optimization problems with deterministic constraints such as latency and the processing time requirement of each request [25]. Goudarzi et al. [29] proposed memetic algorithm to partially/completely process applications' requests on IoT devices or to offload them to edge or cloud servers such that the weighted sum of the application's execution time and device's energy consumption, while processing and communication, is the minimum. However, the memetic algorithm [29] does not assure an optimal solution for problems that involve multiple local minima for total energy consumption [36].

Peng et al. [30] proposed a non-dominated sorting genetic algorithm (NSGA)-II that allows mobile devices to execute applications either locally or offload them to edge or cloud servers such that the device's energy consumption, the application's total execution time, and the cost for using edge/cloud resources are the minimum, with an SLA constraint on execution time. The execution time for local computing includes processing, edge computing includes transmission, waiting, and processing, and cloud computing includes transmission and processing. Xu et al. [22] proposed an energy-aware NSGA-II algorithm that enables a mobile user to offload a request to the nearest access point, i.e., a source point. The request is then transmitted to an edge server-enabled access point, i.e., a destination point, for processing. The algorithm aims to find a path between source and destination points such that the request's offloading time, edge server's energy consumption, and variance in the average edge servers' utilization are the minimum. However, NSGA-II [22,30] suffers from premature convergence similar to the iterative-based algorithm. In addition, the algorithm's convergence rate reduces with increasing objective functions and constraints in the optimization problem [34].

Another offloading algorithm is proposed by Ning et al. [21] proposed a framework to offload vehicles' requests to a group of edge servers in a way that the edge server's energy consumption is the minimum while having an SLA constraint on the requests' deadline. The energy consumption includes processing, transmission among edge servers, and transmission of response back to the vehicle. The execution time involves transmission to an edge server from the vehicle, transmission among edge servers, and processing. Zhai et al. [23] proposed a heuristic-based offloading algorithm to execute requests either locally on a mobile device or offload them to nearby edge devices such that the weighted sum of the request's energy consumption while computing and communication and total execution time is the minimum with an SLA constraint on the total execution time. However, this method is probabilistic and thus cannot assure an optimal solution [37].

3. System model

Fig. 1 shows our edge–cloud integrated computing system model for vehicular networks with bi-directional traffic flow. Our proposed model consists of m vehicles, n requests, o RSUs and edge servers, and p heterogeneous cloud servers. RSUs are placed along the road equidistant from each other. Each RSU_j $(1 \leq j \leq o)$ has a limited coverage range and is equipped with edge servers through wired connections. A vehicle, v_h $(1 \leq h \leq m)$

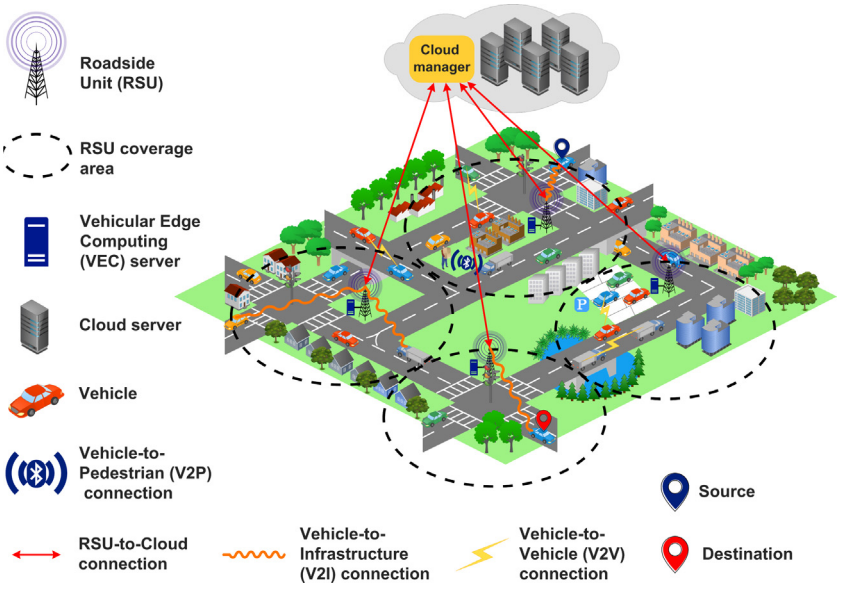


Fig. 1. Edge–cloud integrated computing system model for vehicular networks.

can communicate to an edge server e_j $(1 \leq j \leq o)$ only if it is under the range of RSU_j. Edge servers ensure low latency compared to cloud servers, aiding in real-time processing for time-critical applications, such as traffic alerts, accident prevention, and real-time navigation. This is because of the proximity between vehicles and edge servers compared to that between vehicles and cloud servers. However, the processing and storage capabilities of edge servers are lower compared to that of the cloud servers, making the edge servers bottleneck for compute-intensive applications, such as multimedia, augmented reality, and autonomous driving. Moreover, the edge and cloud servers consume high energy while processing the vehicle's requests. Consequently, it becomes crucial to carefully process requests at edge or cloud server to minimize the energy consumption of the integrated edge–cloud computing system while respecting the requests' SLA requirements. A heterogeneous communication bandwidth between each edge and cloud server is considered.

Each edge server, e_j, in our proposed model consists of scheduling and processing queues, and each cloud server, c_k $(1 \leq k \leq p)$, consists of a processing queue. The scheduling queue is responsible for making the offloading decision for each request, while the processing queue is responsible for executing the request. A vehicle, v_h, submits a request, r_i $(1 \leq i \leq n)$, to the edge server of the RSU under whose communicating range the vehicle is. A request is represented as a tuple $r_i = \{Len_{r_i}, Size_{r_i}, CPU_{r_i}, L^{max}_{r_i}, PT^{max}_{r_i}, Speed_{v_h,r_i}, (x^{src}_{v_h,r_i}, y^{src}_{v_h,r_i}), (x^{des}_{v_h,r_i}, y^{des}_{v_h,r_i})\}$. Each request in our system model is atomic and cannot be further divided. Consequently, each request can be offloaded to at most one edge or cloud server for execution.

A set of requests, when submitted by vehicles, enters the scheduling queue of an edge server. The scheduling queue makes the offloading decision for each request, i.e., whether to execute a request locally on the edge itself or to offload it to one of the cloud servers for execution. The offloading decision is made in a way that the total energy consumption for executing all requests is the minimum and each request's SLA requirements, in terms of latency and processing time, are maintained. If the decision for a request is to execute locally, then the request enters the processing queue of the edge server e_j. If the decision for a request is to be executed by a cloud server c_k, then the scheduling queue of e_j sends the request along with the information of the allocated cloud server c_k to the cloud manager. The cloud manager then submits the request to the processing queue of the corresponding cloud server. The list of notations, used in this paper, and their definitions are listed in Table 2.

4. Problem formulation

A set of n requests corresponding to different vehicular applications, such as autonomous driving, infotainment, augmented reality, accident prevention, traffic alert, is generated by m vehicles. Each request r_i by a vehicle v_h is submitted to the communicating edge server e_j. Depending on the energy consumption and resource utilization of the edge and cloud servers and requests' characteristics, i.e., compute-intensive or time-critical, each request should be scheduled for execution locally at the edge server or offloaded to a cloud server. The scheduling queue of edge server e_j makes an offloading decision such that the total energy consumption of the requests is the minimum and each request maintains the SLA constraints in terms of latency and processing time.

The latency and processing time of r_i when executed either on e_j or c_k are presented in Eqs. (1) and (2) respectively. The latency (Eq. (1)) depends on the following: (1) server on which the request is executed, and (2) position of v_h while receiving the response, res_i, of the request, as the vehicle might be moving.

$$L_{r_i} = \begin{cases} T^{com}_{r_i(v_h,e_j)} + T^{com}_{res_i(e_j,v_h)}, & \text{case (a)} \\ T^{com}_{r_i(v_h,e_j)} + T^{com}_{res_i(e_j,c_k)} + T^{com}_{res_i(c_k,e_{j+y})} + T^{com}_{res_i(e_{j+y},v_h)}, & \text{case (b)} \\ T^{com}_{r_i(v_h,e_j)} + T^{com}_{r_i(e_j,c_k)} + T^{com}_{res_i(c_k,e_{j+y})} + T^{com}_{res_i(e_{j+y},v_h)}, & \text{case (c)} \end{cases} \tag{1}$$

$$PT_{r_i} = \begin{cases} \frac{Len_{r_i}}{S_{e_j}}, & \text{case (a) or case (b)} \\ \frac{Len_{r_i}}{S_{c_k}}, & \text{case (c)} \end{cases} \tag{2}$$

where $T^{com}_{r_i(a,b)}$ represents the communication time required to transmit request r_i from a to b and $T^{com}_{res_i(a,b)}$ represents the communication time required to transmit response of request, i.e., res_i, from a to b.

The different cases are as follows:

- *Case (a):* r_i is executed locally on e_j and v_h is in the range of RSU_j while receiving the response. Consequently, the latency

Table 2
List of notations and their definitions.

Notation	Definition
h, m, v_h	Vehicle index, number of vehicles, hth vehicle
i, n, r_i, res_i	Request index, number of requests, ith request, response of r_i
j, o, RSU_j, e_j	RSU and edge server index, number of RSUs and edge servers, jth RSU, jth edge server
k, p, c_k	Cloud server index, number of cloud servers, kth cloud server
z, s_z	Server (edge/cloud) index, zth server ($where\ s_z \in \{e_j, c_k\}$)
Len_{r_i}	Length of request i in Million Instructions (MI)
$Size_{r_i}$, $Size_{res_i}$	Size of request i, size of response for request i in bits
CPU_{r_i}	CPU utilization of request i
$L_{r_i}^{max}$	Maximum tolerable latency for request i
$PT_{r_i}^{max}$	Maximum tolerable processing time for request i
$Speed_{v_h,r_i}$	Speed of vehicle h when submitting request i
$(x_{v_h,r_i}^{src}, y_{v_h,r_i}^{src})$	Source of vehicle h, in terms of longitude (x_{v_h,r_i}^{src}) and latitude (y_{v_h,r_i}^{src}), while submitting request i
$(x_{v_h,r_i}^{des}, y_{v_h,r_i}^{des})$	Destination of vehicle h, in terms of longitude (x_{v_h,r_i}^{des}) and latitude (y_{v_h,r_i}^{des}), who submitted request i
L_{r_i}	Latency of request i when executed
$T_{r_i(a,b)}^{com}$	Communication time required to transfer request i from a to b ($where\ a \in \{v_h, e_j\}$ and $b \in \{e_j, c_k\}$)
$T_{res_i(a,b)}^{com}$	Communication time required to transfer the response of request i from a to b ($where\ a \in \{e_j, c_k, e_{j+y}\}$ and $b \in \{v_h, c_k, e_{j+y}\}$)
$B_{a,b}$	Communication bandwidth between a to b ($where\ a, b \in \{v_h, e_j, c_k, e_{j+y}\}$)
PT_{r_i}	Processing time of request i
S_{e_j}, S_{c_k}	Processing speed of e_j, processing speed of c_k in terms of Million Instructions per Second (MIPS)
$E_{r_i(s_z)}$	Energy consumed by server s_z while executing request i
$P_{r_i(s_z)}$	Power consumed by server s_z while executing request i
$P'_{CPU_{r_i(s_z)}}$	Power predicted using linear regression for executing request i on server s_z
$\phi_{CPU_{r_i(s_z)}}$	Error correction term for the Locally Corrected Linear Regression (LC-LR) model while executing request i on server s_z
α_{s_z}, β_{s_z}	Intercept, slope for linear regression model
CPU'_{s_z}, CPU'_{s_z}	Lower, upper CPU utilization values of server s_z such that $CPU'_{s_z} \leq CPU_{r_i} \leq CPU''_{s_z}$
$e_{CPU'_{s_z}}$ and $e_{CPU''_{s_z}}$	Error between the actual and predicted (using linear regression) power consumption values for CPU utilization CPU' and CPU'' on server s_z
pop_{size}	Size of the population, i.e., the subset of offloading solutions considered in each generation of genetic algorithm
l	Offloading solution index
F'_l, $\tilde{F}'_l$, F^a_l, F_l	Non-penalized, normalized non-penalized, adaptive, penalized fitness scores of offloading solution l
v_l, $\tilde{v}_l$	Constraints violation, normalized constraints violation for offloading solution l
$LV_{r_i}(l)$	Latency violation of request i for offloading solution l
$PTV_{r_i}(l)$	Processing time violation of request i for offloading solution l
v_l^{lat}	Total latency violation of all requests for offloading solution l
v_l^{proc}	Total processing time violation of all requests for offloading solution l
F'_{min}, F'_{max}	Minimum, maximum non-penalized fitness scores among all offloading solutions in the current subset of solutions
v_{max}^{lat}	Maximum latency violation among all solutions in the current subset of solutions
v_{max}^{proc}	Maximum processing time violation among all offloading solutions in the current subset of solutions
n_f, r_f	Number, fraction of feasible offloading solutions in the current subset of solutions
fp_l, cp_l	Fitness probability, cumulative fitness probability for offloading solution l
μ_c, μ_m	Crossover rate, mutation rate
n_{mut}	Number of requests for which the server allocation should be changed in the current subset of solutions
l_c	Length of an offloading solution that is equal to the number of requests

involves the communication time to send r_i from the v_h to e_j, and the communication time to send res_i from e_j to v_h.

- *Case (b):* r_i is executed locally on e_j, but v_h is not in the range of RSU_j while receiving the response. v_h will be in the range of RSU_{j+y} which represents yth RSU after RSU_j on the path between the v_h's source and destination. The value of y can be determined based on $Speed_{v_h,r_i}$, $(x_{v_h,r_i}^{src}, y_{v_h,r_i}^{src})$ and $(x_{v_h,r_i}^{des}, y_{v_h,r_i}^{des})$ [31]. In this case, the latency involves the communication time to send r_i from v_h to e_j and the time to send res_i from e_j to cloud server c_k, from c_k to e_{j+y}, and from e_{j+y} to v_h.
- *Case (c):* r_i is offloaded to one of the cloud servers, c_k for execution. Consequently, the latency involves the time to send r_i from v_h to e_j and then from e_j to c_k, and the time to send res_i from c_k to e_{j+y} and from e_{j+y} to v_h.

The communication times presented in Eq. (1) can be calculated by using Eqs. (3)–(8).

$$T_{r_i(v_h,e_j)}^{com} = \frac{Size_{r_i}}{B_{v_h,e_j}} \tag{3}$$

$$T_{res_i(e_j,v_h)}^{com} = \frac{Size_{res_i}}{B_{e_j,v_h}} \tag{4}$$

$$T_{res_i(e_j,c_k)}^{com} = \frac{Size_{res_i}}{B_{e_j,c_k}} \tag{5}$$

$$T_{res_i(c_k,e_{j+y})}^{com} = \frac{Size_{res_i}}{B_{c_k,e_{j+y}}} \tag{6}$$

$$T_{res_i(e_{j+y},v_h)}^{com} = \frac{Size_{res_i}}{B_{e_{j+y},v_h}} \tag{7}$$

$$T_{r_i(e_j,c_k)}^{com} = \frac{Size_{r_i}}{B_{e_j,c_k}} \tag{8}$$

The energy consumption of r_i when processed on server s_z ($where\ s_z \in \{e_j, c_k\}$) can be computed using Eq. (9).

$$E_{r_i(s_z)} = P_{r_i(s_z)} \times PT_{r_i} \tag{9}$$

To estimate the power consumed by s_z while processing r_i, the Locally Corrected Linear Regression (LC-LR) power model is used as it outperformed other power models in our previous work [38]. LC-LR is an extension of the classical linear regression model ($P'_{CPU_{r_i(s_z)}}$) by adding an error correction term as stated in Eqs. (10) and (11). In this paper, a CPU-based linear regression power model is used because CPU is considered to be the most dominant power-consuming resource in a computing server [39].

$$P_{r_i(s_z)} = P'_{CPU_{r_i(s_z)}} + \phi_{CPU_{r_i(s_z)}} \tag{10}$$

$$P'_{CPU_{r_i(s_z)}} = \alpha_{s_z} + [\beta_{s_z} \times CPU_{r_i}] \tag{11}$$

The regression coefficients α_{s_z} and β_{s_z} are server-specific and their values should be obtained experimentally for each server. This is using a training dataset consisting of server's CPU utilization values at different intervals and the corresponding power consumptions. The error correction term can be calculated by constructing a linear model between the CPU utilization values CPU'_{s_z} and CPU''_{s_z}, such that $CPU'_{s_z} \leq CPU_{r_i} \leq CPU''_{s_z}$, as stated in Eq. (12). The term $e_{CPU'_{s_z}}$ represents the intercept of the model and $\frac{e_{CPU''_{s_z}} - e_{CPU'_{s_z}}}{CPU''_{s_z} - CPU'_{s_z}}$ represents the slope.

$$\phi_{CPU_{r_i(s_z)}} = e_{CPU'_{s_z}} + \frac{(e_{CPU''_{s_z}} - e_{CPU'_{s_z}})(CPU_{r_i} - CPU'_{s_z})}{(CPU''_{s_z} - CPU'_{s_z})} \tag{12}$$

There are multiple ways that each edge server can schedule and offload a set of received requests. This paper focuses on finding the optimal solution for computation offloading in edge–cloud integrated vehicular networks which minimizes the total energy consumption of all the requests. The offloading decision should be made in a way that the latency and processing time requirements of each request should not be violated. Hence, the considered optimization problem is formulated as follows.

Problem:

$$\text{Minimize} \sum_{i=1}^{n} E_{r_i(s_z)},\ s_z \in \{e_j, c_k\} \tag{13}$$

Subject to:

$$L_{r_i} \leq L_{r_i}^{max},\ \forall i \in \{1, 2, \ldots, n\} \tag{14}$$

$$PT_{r_i} \leq PT_{r_i}^{max},\ \forall i \in \{1, 2, \ldots, n\} \tag{15}$$

Here $\sum_{i=1}^{n} E_{r_i(s_z)}$ represents the total energy consumption of all the requests in the system. The constraint in Eq. (14) ensures that the latency of each request in the system is not more than the tolerable latency. Constraint in Eq. (15) ensures that the processing time requirement of each request is satisfied.

Computation offloading is an NP-hard problem where the search space and time to optimal solution increase exponentially with the number of requests required to be offloaded. Consequently, in this paper, EGA is adopted to obtain a near-optimal solution in a polynomial time.

5. Proposed algorithm

In this section, EGA [32] with adaptive fitness is proposed to find a solution of the optimization problem (Eqs. (13)–(15)). EGA, inspired by Charles Darwin's theory of natural evolution, is an iterative process that explores multiple nodes in the search space simultaneously. In the context of the considered optimization problem, a node in the search space corresponds to an offloading solution for a set of requests. The subset of offloading solutions in each iteration is known as population, the number of offloading solutions in the subset represents the population size, and the population in each iteration is known as a generation. The population size remains constant throughout the generations. Each offloading solution is represented by a sequence of request-server allocation tuple, (r_i, e_j) or (r_i, c_k). Each tuple is known as a gene and the sequence of genes is known as a chromosome. The length of a chromosome, l_c, is equal to the number of requests (n). Fig. 2 shows an example of a chromosome that represents an offloading solution for 5 requests, i.e., ($l_c = 5$), where r_1 and r_4 are executed locally on e_1, r_2 is offloaded to c_2 for execution, and r_3 and r_5 are offloaded to c_1. EGA resembles the process of natural selection where the fittest offloading solutions, i.e., the ones near the optimal solution, are selected from the subset to produce offloading solutions for the next generation. The objective function (Eq. (13)) is used to determine the fitness of offloading solutions. To incorporate the inequality constraints on latency (Eq. (14)) and processing time (Eq. (15)), an adaptive fitness approach is implemented that penalizes the offloading solutions with SLA violations by reducing their fitness.

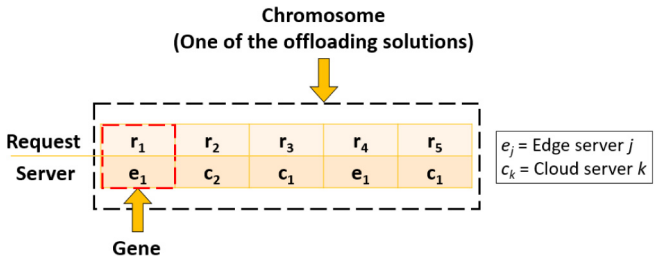


Fig. 2. Example of a chromosome that represents one of the possible offloading solutions.

The proposed algorithm consists of 6 stages: (1) initialization of a subset of offloading solutions, (2) evaluation of the solutions in the subset, (3) selection of the fittest solutions, (4) crossover to produce offspring solutions, (5) mutation of server allocations for some requests, and (6) termination. In the first stage, an initial subset of offloading solutions is randomly generated (i.e., initial population), where a solution consists of server allocations for the requests. Each solution in the subset is evaluated in the second stage to determine how close it is to the optimal solution. This is using an adaptive fitness function based on the optimization objective and constraints stated in Eqs. (13)–(15). The fit solutions in the subset, i.e., the ones close to the optimal solution, are then selected using a probabilistic approach in the third stage. In the fourth stage, the selected solutions are used to produce offspring offloading solutions leading to convergence. Server allocations of some requests are randomly changed in the fifth stage to avoid premature convergence. The algorithm terminates in the last stage based on the termination conditions. Steps 2–6 are iterated until termination. Each stage is explained in detail in the following subsections using a numerical example.

Example. A vehicular network consisting of 1 edge server, 2 cloud servers, and 5 vehicles. Each vehicle submits a request to the edge server. For each request, the edge server should decide to execute it locally or to offload it to one of the cloud servers. The offloading decision is made in a way that the total energy consumption of the requests is the minimum considering the latency and processing time constraints for each request.

Solution 1	r_1	r_2	r_3	r_4	r_5
	c_2	c_1	c_2	e_1	e_1
Solution 2	r_1	r_2	r_3	r_4	r_5
	e_1	e_1	c_2	c_2	c_2
Solution 3	r_1	r_2	r_3	r_4	r_5
	e_1	e_1	e_1	e_1	c_2
Solution 4	r_1	r_2	r_3	r_4	r_5
	e_1	c_1	c_2	c_1	e_1
Solution 5	r_1	r_2	r_3	r_4	r_5
	c_1	e_1	c_1	c_1	c_2

Fig. 3. A random subset of offloading solutions at the initialization stage for the considered example.

5.1. Initialization: Subset of offloading solutions

In this stage, an initial subset of offloading solutions is randomly generated, i.e., each request is randomly allocated either to e_j or c_k for execution. The number of solutions in the subset is determined by the parameter pop_{size}. The value of pop_{size} should be carefully selected as a small value may result in faster convergence but the solution might get trapped in local optima, while a large value increases the search space with slower convergence towards global optima. The pseudocode for initialization of a subset of offloading solutions is presented in Algorithm 1. Fig. 3 shows the randomly generated offloading solutions for the considered example.

Algorithm 1 Initialization of a subset of offloading solutions

Input: pop_{size}, requests r_i $(1 \leq i \leq n)$, edge servers e_j $(1 \leq j \leq o)$, cloud servers c_k $(1 \leq k \leq p)$

Output: Initial subset of offloading solutions

```
l ← 1                                          ▷ initialize offloading solution index
while l ≤ pop_size do                          ▷ for each solution in the population
    for i = 1 to n do                          ▷ for each request in solution l
        request ← r_i                          ▷ determine the request
        server ← selectRandom ∈ {e_j, c_k}     ▷ randomly select edge server e_j or cloud server c_k
    end for
    offloading_solution(l) ← tuple(request,server)   ▷ assign server allocation to l
end while
```

5.2. Evaluation: Input offloading solutions

In this stage, each solution from the initial subset of offloading solutions is evaluated in terms of fitness. Each solution has an associated fitness score, calculated using a fitness function, that represents how near the solution is from the optimal solution. In the proposed algorithm, the energy optimization objective (Eq. (13)) is used to define the fitness function. Moreover, to handle the optimization constraints (Eqs. (14) and (15)), a penalty function is used that penalizes the infeasible offloading solutions, i.e., the ones for which the constraints are violated. The general representation of a penalized fitness function is represented in Eq. (16), where the penalty function is added (subtracted) to (from) the non-penalized fitness score. In this paper, the penalty function is added to the non-penalized fitness score, to avoid a negative fitness score value.

$$F_l^a = F_l' \pm v_l, \;\; l = \{1, 2, \ldots, pop_{size}\} \tag{16}$$

Different methods based on penalty functions have been proposed in the literature: death, static, dynamic, annealing, co-evolutionary, and adaptive [40]. In this paper, the adaptive penalty method [41] is used due to the following advantages of the adaptive method over other methods: (1) no parameter tuning is required for the penalty function, (2) computationally less expensive, (3) easy to implement, and (4) works well even if the feasible search space is very small compared to the entire search space.

In the proposed algorithm, an optimal solution represents server allocation for requests in a way that the total energy consumption of the requests is the minimum and no request violates the SLA. This implies that the lower the energy consumption and SLA violations for a solution, the closer the solution would be to the optima and higher the fitness score. Consequently, to assign the highest fitness score to the optimal solution, the reciprocal of the sum of non-penalized fitness score (i.e., energy consumption) and penalty factor (i.e., SLA violations) is calculated. The fitness score for each offloading solution using an adaptive penalty function can be computed as follows:

- Compute the non-penalized fitness score for each solution in the current subset of solutions using Eq. (17).

$$F_l' = \sum_{i=1}^{n} E_{r_i(s_z)} \tag{17}$$

- Compute latency and processing time violations, for each request in each solution, using Eqs. (18) and (19).

$$LV_{r_i}(l) = \begin{cases} L_{r_i}(l) - L_{r_i}^{max}, & L_{r_i}^{max} < L_{r_i}(l) \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

$$PTV_{r_i}(l) = \begin{cases} PT_{r_i}(l) - PT_{r_i}^{max}, & PT_{r_i}^{max} < PT_{r_i}(l) \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

- Compute total latency and processing time violations for all requests in each solution using Eqs. (20) and (21).

$$v_l^{lat} = \sum_{i=1}^{n} LV_{r_i}(l) \tag{20}$$

$$v_l^{proc} = \sum_{i=1}^{n} PTV_{r_i}(l) \tag{21}$$

- Normalize the non-penalized fitness score and the constraints violations, i.e., the penalty factor, for each offloading solution using Eqs. (22) and (23).

$$\tilde{F}_l' = \frac{F_l' - F_{min}'}{F_{max}' - F_{min}'} \tag{22}$$

$$\tilde{v}_l = \frac{1}{2}\left(\frac{v_l^{lat}}{v_{max}^{lat}} + \frac{v_l^{proc}}{v_{max}^{proc}}\right) \tag{23}$$

- Compute the adaptive fitness for the feasible and infeasible offloading solutions using Eq. (24).

$$F_l^a = \begin{cases} \tilde{v}_l, & n_f = 0 \\ \tilde{F}_l', & \tilde{v}_l = 0 \\ \sqrt{(\tilde{F}_l')^2 + (\tilde{v}_l)^2} + [(1 - r_f)\tilde{v}_l + (r_f)\tilde{F}_l'], & \text{otherwise} \end{cases} \tag{24}$$

where r_f is the fraction of feasible offloading solutions in the subset of solutions as stated in Eq. (25).

$$r_f = \frac{n_f}{pop_{size}} \tag{25}$$

Non-penalized fitness score	Latency violation	Processing time violation	Normalized non-penalized fitness score	Normalized constraints violation	Adaptive fitness score	Penalized fitness score
$F'_1 = 3457.741$	$v_1^{lat} = 0$	$v_1^{proc} = 0.547$	$\tilde{F}'_1 = 0.494$	$\tilde{v}_1 = 0.215$	$F_1^a = 0.215$	$F_1 = 0.823$
$F'_2 = 1733.655$	$v_2^{lat} = 0$	$v_2^{proc} = 0.725$	$\tilde{F}'_2 = 0$	$\tilde{v}_2 = 0.285$	$F_2^a = 0.285$	$F_2 = 0.778$
$F'_3 = 2090.85$	$v_3^{lat} = 0$	$v_3^{proc} = 1.272$	$\tilde{F}'_3 = 0.102$	$\tilde{v}_3 = 0.5$	$F_3^a = 0.5$	$F_3 = 0.667$
$F'_4 = 5220.491$	$v_4^{lat} = 0$	$v_4^{proc} = 0.205$	$\tilde{F}'_4 = 1$	$\tilde{v}_4 = 0.080$	$F_4^a = 0.080$	$F_4 = 0.926$
$F'_5 = 4977.242$	$v_5^{lat} = 0$	$v_5^{proc} = 0.52$	$\tilde{F}'_5 = 0.930$	$\tilde{v}_5 = 0.204$	$F_5^a = 0.204$	$F_5 = 0.30$

Fig. 4. Evaluation of the offloading solutions for the considered example.

- Compute the penalized fitness score for each offloading solution as the reciprocal of the adaptive fitness score using Eq. (26). To avoid division by zero, 1 is added to the denominator.

$$F_l = \frac{1}{F_l^a + 1} \tag{26}$$

The pseudocode for evaluating the subset of offloading solutions is presented in Algorithm 2. Fig. 4 shows the fitness evaluation of the randomly generated offloading solutions (Fig. 3) for the considered example.

Algorithm 2 Evaluation of the subset of offloading solutions

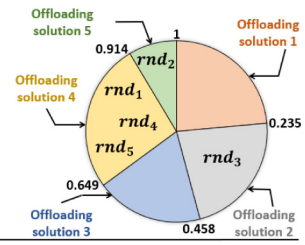
Input: Initial subset of random offloading solutions, requests r_i $(1 \leq i \leq n)$
Output: Fitness score of each offloading solution from the subset

for $l = 1$ to pop_{size} **do** ▷ for each solution l
  $F_l^{'} \leftarrow$ Equation (17) ▷ compute non-penalized fitness
  for $i = 1$ to n **do** ▷ for each r_i in l
    $LV_{r_i}(l) \leftarrow$ Equation (18) ▷ compute latency violation
    $PTV_{r_i}(l) \leftarrow$ Equation (19) ▷ compute processing time violation
  end for
  $v_l^{lat} \leftarrow$ Equation (20) ▷ compute total latency violation
  $v_l^{proc} \leftarrow$ Equation (21) ▷ compute total processing time violation
  $\tilde{F}_l^{'} \leftarrow$ Equation (22) ▷ compute normalized non-penalized fitness
  $\tilde{v}_l \leftarrow$ Equation (23) ▷ compute normalized violations
  $F_l^a \leftarrow$ Equation (24) ▷ compute adaptive fitness
  $F_l \leftarrow$ Equation (26) ▷ compute penalized fitness
end for

5.3. Selection: Fittest offloading solutions

In this stage, offloading solutions from the subset are selected based on their penalized fitness score to produce offspring solutions. The proposed algorithm uses Roulette Wheel Selection (RWS) method for selection as it is the most used method in the context of requests' scheduling in computing systems [42]. RWS is a fitness proportionate selection method where the selection probability for an offloading solution is proportional to the solution's fitness. In RWS, the fitness probability and cumulative probability of each offloading solution in the subset are calculated using Eqs. (27) and (28) respectively. A roulette wheel is constructed using the computed probabilities such that the area occupied by each solution is proportionate to its fitness probability. A random number for each solution is then generated and placed under the area on the roulette wheel where it belongs. The solutions under whose area the random numbers belong are selected as the new solutions. The pseudocode for selecting the fittest solutions is presented in Algorithm 3.

Fitness probability	Cumulative probability	Random number
$fp_1 = 0.235$	$cp_1 = 0.235$	$rnd_1 = 0.8035$
$fp_2 = 0.223$	$cp_2 = 0.458$	$rnd_2 = 0.9569$
$fp_3 = 0.191$	$cp_3 = 0.649$	$rnd_3 = 0.2921$
$fp_4 = 0.265$	$cp_4 = 0.914$	$rnd_4 = 0.8549$
$fp_5 = 0.086$	$cp_5 = 1$	$rnd_5 = 0.8606$



Selected solution 1 = Solution 4

r_1	r_2	r_3	r_4	r_5
e_1	c_1	c_2	c_1	e_1

Selected solution 2 = Solution 5

r_1	r_2	r_3	r_4	r_5
c_1	e_1	c_1	c_1	c_2

Selected solution 3 = Solution 2

r_1	r_2	r_3	r_4	r_5
e_1	e_1	c_2	c_2	c_2

Selected solution 4 = Solution 4

r_1	r_2	r_3	r_4	r_5
e_1	c_1	c_2	c_1	e_1

Selected solution 5 = Solution 4

r_1	r_2	r_3	r_4	r_5
e_1	c_1	c_2	c_1	e_1

Fig. 5. Selection of the fittest offloading solutions using the Roulette Wheel Selection method for the considered example.

$$fp_l = \frac{F_l}{\sum_{l=1}^{pop_{size}} F_l} \tag{27}$$

$$cp_l = \sum_{i=1}^{l} fp_l \tag{28}$$

Algorithm 3 Selection of the fittest solutions

Input: Initial subset of random offloading solutions, pop_{size}
Output: Subset of solutions after selection

for $l = 1$ to pop_{size} **do** ▷ for each solution l
  $fp_l \leftarrow$ Equation (27) ▷ compute fitness probability for l
  $cp_l \leftarrow$ Equation (28) ▷ compute cumulative probability for l
  $rnd(l) \leftarrow$ RandomNumber $\in U[0, 1]$ ▷ generate a random number for l
end for
for $k = 1$ to pop_{size} **do** ▷ for each solution k
  for $l = 1$ to pop_{size} **do** ▷ for each solution l
    if $rnd(k) > cp_l$ **then** ▷ determine the position of random number for k on the roulette wheel
      selected_solution(k) $\leftarrow$ solution(l-1) ▷ replace solution k by l
    end if
  end for
end for

Fig. 5 depicts the selection of the fittest offloading solutions using RWS for the considered example. It shows that the random number generated for solution 1 lies under the area of solution 4 on the roulette wheel. Consequently, the selected solution 1 is solution 4 from the initial subset of offloading solutions. Similarly, the selected solutions 2, 3, 4, and 5 are the 5th, 2nd, 4th, and 4th solutions respectively, from the initial subset of solutions. As solution 4 in the initial subset of solutions was the fittest, i.e., having the least SLA violations (Fig. 4), it has been selected the majority of the times by the RWS method.

5.4. Crossover: Offspring offloading solutions reproduction

In this stage, a crossover operation is performed to converge the algorithm towards the optimal offloading solution. It improves the performance of the algorithm by exploiting the search space within the neighborhood of fit solutions that are selected from the previous stage. It increases the number of fit offloading solutions in the subset of solutions by swapping the server allocations between two selected offloading solutions, referred to as parent solutions. The number of parent solutions selected for crossover operation depends on the parameter μ_c. For instance, $\mu_c = 0.5$ indicates that approximately 50% of the solutions will be selected at random for the crossover. In this paper, a single-point crossover is used in which the server allocations between the parents are swapped for all the requests after a randomly generated cutoff point. The crossover operation begins by generating uniform random numbers between 0 and 1 for each solution in the subset. The solutions, for which the value of the random number is less than the μ_c, are selected for crossover. The selected solutions are randomly paired. For each pair of solutions, i.e., parent 1 and parent 2, a cutoff point value is generated randomly to decide for which requests the allocations should be swapped. A crossover operation between parent solutions will generate two offspring solutions. The parent solutions in the population are then replaced with the top two fittest solutions among the parents and the offspring. Consequently, a fitter subset of offloading solutions is used in the next generation. The pseudocode for the crossover operation is presented in Algorithm 4.

Algorithm 4 Crossover operation to produce offspring offloading solutions

Input: pop_{size}, μ_c, l_c
Output: Subset of offloading solutions after crossover

for $l = 1$ to pop_{size} **do** ▷ for each solution l
  $rnd(l) \leftarrow$ RandomNumber $\in U[0, 1]$ ▷ generate a random number for l
  if $rnd(l) < \mu_c$ **then**
    selected_solution_for_crossover $\leftarrow l$ ▷ select a solution whose random number is less than crossover rate
  end if
end for
pairs $\leftarrow$ GeneratePairs(selected_solution_for_crossover) ▷ generate pairs from selected solutions
for each *pair* $\in$ *pairs* **do**
  select$parent\_solution_1$ *and* $parent\_solution_2$ *from pair* ▷ assign solutions in each pair as parents
  cutoff $\leftarrow$ RandomNumber $\in U[1, l_c - 1]$ ▷ randomly generate a cutoff point
  $k \leftarrow 1$
  for $1 \leq k \leq cutoff$ **do** ▷ for all requests till the cutoff
    $offspring\_solution_1(k) \leftarrow parent\_solution_1$ ▷ assign server allocation of parent 1 to offspring 1
    $offspring\_solution_2(k) \leftarrow parent\_solution_2$ ▷ assign server allocation of parent 2 to offspring 2
  end for
  for $cutoff < k \leq l$ **do** ▷ for all requests after the cutoff
    $offspring\_solution_1(k) \leftarrow parent\_solution_2$ ▷ assign server allocation of parent 2 to offspring 1
    $offspring\_solution_2(k) \leftarrow parent\_solution_1$ ▷ assign server allocation of parent 1 to offspring 2
  end for
  $solution(pair) \leftarrow Best(parent\_solution_1, parent\_solution_2, offspring\_solution_1, offspring\_solution_2)$ ▷ two fit solutions
end for

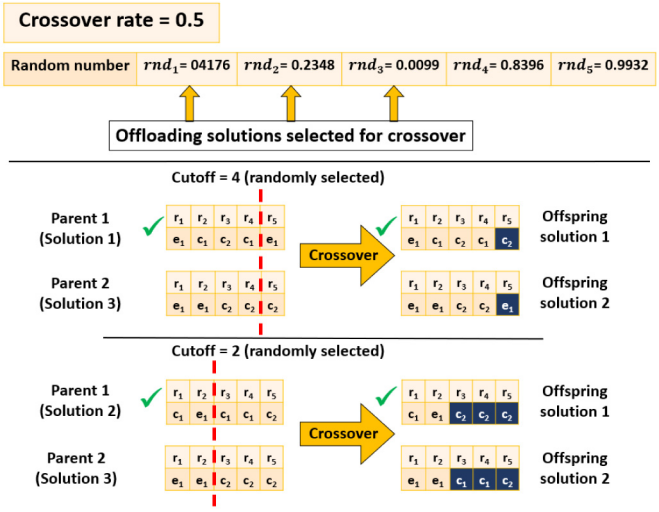


Fig. 6. Crossover operation on the offloading solutions to produce offspring solutions for the considered example.

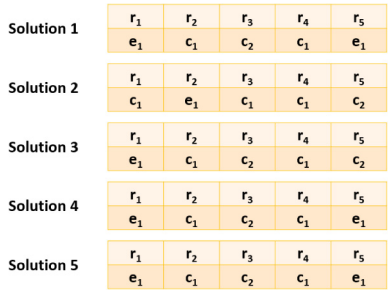


Fig. 7. Offloading solutions after crossover for the considered example.

Fig. 6 shows the crossover operation performed on the offloading solutions (Fig. 5) for the considered example, where solutions 1, 2, and 3 are selected for crossover. Based on the selected parent solutions, two pairs are formed for crossover, i.e., solutions 1 – 3 and solutions 2 – 3. Cutoff values of 4 and 2 are generated randomly for first and second pairs, respectively. Two offspring solutions are generated for the first pair by swapping the allocations of the parent solutions after the cutoff (i.e., the allocation for request 5). Similarly, for the second pair, the allocations for the requests after the cutoff value of 2 (i.e., requests 3, 4, and 5) are swapped. The top two offloading solutions from the first pair of the parents and the generated offspring solutions are parent 1 and offspring 1. Solution 1 and solution 3 from the subset of solutions are then replaced by these top solutions. Similarly, the parent solutions for the second pair are replaced by the top two solutions among the parents and their offspring solutions. The new subset of solutions after the crossover operation is shown in Fig. 7.

5.5. Mutation: Server allocations of some requests

In this stage, offloading solutions from the subset of solutions are diversified by performing the mutation operation. Mutation explores the search space by randomly changing the server allocations for some requests from the subset of solutions. This helps the algorithm in escaping the local optima, i.e., avoids premature convergence, and increases the probability of finding the global optima. Consequently, in contrast to crossover, mutation leads to solutions outside the neighborhood of offloading solutions.

The number of mutations, i.e., the number of requests for which server allocations are changed, can be calculated using Eq. (29). It is controlled by the parameter μ_m. The value of μ_m is kept low, as a high value might prevent the solution to converge to an optimum solution. Requests are selected randomly for reallocation and the server allocations for these requests are changed randomly to perform the mutation operation. The pseudocode for the mutation operation is presented in Algorithm 5.

$$n_{mut} = l_c \times pop_{size} \times \mu_m \tag{29}$$

Algorithm 5 Mutation operation for server allocations of requests

Input: pop_{size}, μ_m, l_c, edge servers e_j $(1 \leq j \leq o)$, cloud servers c_k $(1 \leq k \leq p)$

Output: A mutated subset of offloading solutions

$n_{allocations} \leftarrow l_c \times pop_{size}$ ▷ compute total number of requests allocation in the population
$n_{mut} \leftarrow n_{allocations} \times \mu_m$ ▷ compute number of requests for which the allocations should be mutated
for $z = 1$ to n_{mut} **do**
  $rnd(z) \leftarrow$ RandomNumber $\in U[1, n_{allocations}]$ ▷ generate random number to select a request for mutation
  $rem_z \leftarrow$ remainder after dividing rnd(z) by l_c
  if $rem_z = 0$ **then** ▷ check whether the random number is at the end of solution
    $solution_z \leftarrow \frac{rnd(z) - rem_z}{l_c}$ ▷ determine the solution to which the random number belongs
    $request\_to\_reallocate_z \leftarrow l_c$ ▷ determine the request in the solution for mutation
  else
    $solution_z \leftarrow \left(\frac{rnd(z) - rem_z}{l_c}\right) + 1$
    $request\_to\_reallocate_z \leftarrow rem_z$
  end if
  $request\_to\_reallocate_z, solution_z \leftarrow$ RandomServer $\in \{e_j, c_k\}$ ▷ change the allocation for the selected request
end for

Fig. 8 shows the subset of offloading solutions before and after the mutation operation for the considered example. Three random numbers, 2, 4, and 8 are generated as the number of mutations is $2.5 \approx 3$ (i.e., $5 \times 5 \times 0.1$). Consequently, the server allocations for the 2nd request of solution 1, the 4th request of solution 1, and the 3rd request of solution 2 are then changed. Fig. 9 shows the evaluation of the offloading solutions after the selection, crossover, and mutation operations. It shows that the fitness score of the subset is improved compared to that of the initial subset of solutions (Fig. 4).

5.6. Termination

In this stage, the algorithm is terminated if one of the two conditions is met: (1) the maximum number of generations is reached, or (2) the threshold fitness score is reached.

The pseudocode of the proposed approach is presented in Algorithm 6.

6. Performance evaluation

In this section, the experimental environment and the set of experiments performed to evaluate the proposed algorithm are described. In addition, the experimental results are analyzed, and insights and rationales on the obtained results are presented.

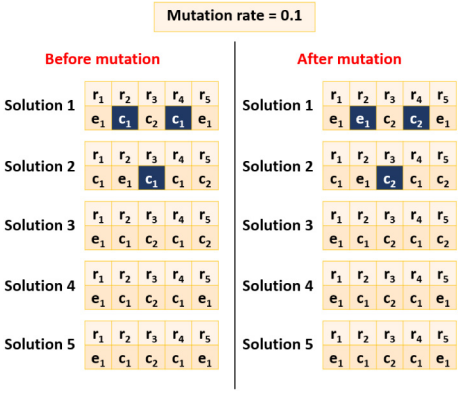


Fig. 8. Subset of offloading solutions before and after mutation operation for the considered example.

Non-penalized fitness score	Latency violation	Processing time violation	Normalized non-penalized fitness score	Normalized constraints violation	Adaptive fitness score	Penalized fitness score
$F'_1 = 1975.262$	$v_1^{lat} = 0$	$v_1^{proc} = 0.725$	$\hat{F}'_1 = 0$	$\hat{v}_1 = 0.5$	$F_1^a = 0.5$	$F_1 = 0.667$
$F'_2 = 4769.764$	$v_2^{lat} = 0$	$v_2^{proc} = 0.52$	$\hat{F}'_2 = 0.861$	$\hat{v}_2 = 0.358$	$F_2^a = 0.358$	$F_2 = 0.736$
$F'_3 = 4978.884$	$v_3^{lat} = 0$	$v_3^{proc} = 0.20$	$\hat{F}'_3 = 0.925$	$\hat{v}_3 = 0.141$	$F_3^a = 0.141$	$F_3 = 0.876$
$F'_4 = 5220.491$	$v_4^{lat} = 0$	$v_4^{proc} = 0.20$	$\hat{F}'_4 = 1$	$\hat{v}_4 = 0.141$	$F_4^a = 0.141$	$F_4 = 0.876$
$F'_5 = 5220.491$	$v_5^{lat} = 0$	$v_5^{proc} = 0.20$	$\hat{F}'_5 = 1$	$\hat{v}_5 = 0.141$	$F_5^a = 0.141$	$F_5 = 0.876$

Fig. 9. Evaluation of the subset of offloading solutions after mutation for the considered example.

Algorithm 6 Energy-SLA-aware evolutionary genetic algorithm-based computation offloading

Input: pop_{size}, requests r_i $(1 \leq i \leq n)$, edge servers e_j $(1 \leq j \leq o)$, cloud servers c_k $(1 \leq k \leq p)$, μ_c, μ_m, l_c

Output: Scheduled requests after offloading

Initialize a subset of offloading solutions using **Algorithm 1** ▷ stage 1
repeat
  Evaluate the solutions in the subset using **Algorithm 2** ▷ stage 2
  Select the fittest solutions using **Algorithm 3** ▷ stage 3
  Perform crossover operation on the selected solutions ↪ to produce offspring solutions using ↪ **Algorithm 4** ▷ stage 4
  Perform mutation operation for reallocating some ↪ requests using **Algorithm 5** ▷ stage 5
until the termination condition is satisfied ▷ stage 6

6.1. Experimental environment

A heterogeneous simulated edge–cloud integrated vehicular network is created consisting of 2 edge servers and 4 cloud servers. The specifications of the servers are presented in Table 3. Servers 1 and 3 (one edge and one cloud server) are part of our Intelligent Distributed Computing and Systems (INDUCE) research laboratory at the College of Information Technology of the United Arab Emirates University. The specifications of the other servers, i.e., servers 2, 4, 5, and 6 are taken from the SPEC Power benchmark suite [43]. The selection of the servers from the SPEC Power benchmark is done in a way that they belong to the same family of the servers present in the laboratory, but with distinct architectures and resource capabilities. The network is implemented using MATLAB.

To develop the LC-LR model for each server used in the experiments (Table 3), a training dataset consisting of the CPU

Table 3
Specifications of the servers used in the experiments.

Location	Server	Specification
Edge	1	Intel Xeon, 2.80 GHz, 2-Core, 512 kB Cache
	2	Intel Xeon E5-2670, 2.60 GHz, 8-Core, 20 MB L3 Cache [44]
Cloud	3	AMD Opteron 252, 2.59 GHz, 2-Core, 1 MB Cache
	4	AMD Opteron 6276, 2.30 GHz, 16-Core [45]
	5	Intel Xeon E5-2699 v3, 2.30 GHz, 18-Core, 45 MB L3 Cache [46]
	6	AMD Opteron 6238 CPU, 2.60 GHz, 12-Core, 16 MB L3 Cache [47]

utilization values of a server at different intervals and the corresponding power consumption values is used. The datasets for servers 1 and 3 are generated in the Laboratory and those for servers 2, 4, 5, and 6 are obtained from the SPEC Power benchmarking results. For the servers in the laboratory, CPU Load Generator [48] is used to stress each server at different CPU utilization values and measure the utilization and corresponding power consumption values. To measure the servers' power consumption, a 4-channel digital oscilloscope, Tektronix – TBS2000 (100 MHz), with 1 GS/s of sampling [49] is used. Each server's power chord is connected to the oscilloscope using voltage and current probes [39] to measure the server's voltage and current respectively. The voltage and current values from the oscilloscope are extracted to a file using a LabVIEW program, and the power consumption is then computed as the product of voltage and current.

To get the positions of the vehicles in our simulated network, the Vehicle–Crowd Interaction (VCI) – DUT dataset [50] is used. The *x_est* and *y_est* columns of the dataset (representing the estimated position of the vehicles) are used as the source location of the vehicles in our experiments. Each vehicle in our experiments generates a static request when the simulation begins. For each request, the proposed algorithm decides on whether to execute it locally on the edge server to which it is submitted or to offload it to a cloud server. The characteristics of the requests are based on different vehicular applications namely face recognition and object detection for autonomous driving, augmented reality, VANET-based health monitoring, and infotainment [51–53]. Table 4 shows the list of different parameters used in the experiments along with their values. The optimal values for the optimization parameters, i.e., crossover rate, mutation rate, and population size, are obtained by evaluating the algorithm with different values of these parameters (Table 4). The value of the termination condition is selected in a way that the algorithm converges before termination. In addition, the algorithm involves two dependent parameters, i.e., length of chromosome and number of mutation, and four random parameters for selection of solutions using RWS, selection of parent solutions for crossover, generation of cutoff values for crossover, and selection of requests for mutation. The value of chromosome length is the same as the number of requests and the value of mutation is calculated based on chromosome length, population size, and mutation rate (Eq. (29)).

6.2. Experiments

In this section, the experiments performed to obtain the training dataset for servers 1 and 3 for power model development and to simulate the vehicular network for implementing our proposed algorithm are explained.

To get the training dataset consisting of CPU utilization values and corresponding power consumptions for servers 1 and 3, each server's CPU is stressed, using a CPU load generator, with CPU utilization between 0% and 100% at an interval of 10%. For each utilization, the CPU is stressed for five minutes, and the server's real-time CPU utilization and power consumption values

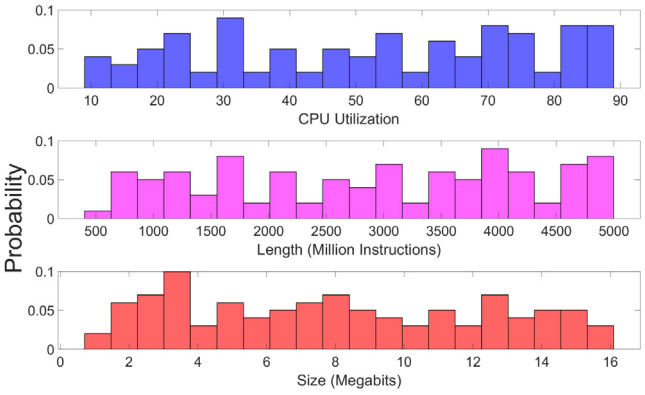


Fig. 10. Probability distribution of the generated requests' CPU utilization, length, and size values.

are measured every second. The measured values are written to a file. The values over five minutes are then averaged. The experiment for each CPU load is repeated five times and all the average values are averaged. Table 5 shows the CPU utilization values and the corresponding power consumption for servers 1–6. CPU utilization is the value returned by a server's operating system which indicates the percentage of total processing power given to a process at a certain time.

To evaluate the performance of our proposed algorithm, a heterogeneous edge–cloud integrated vehicular network is simulated with increasing vehicles. For the geographical location of each server, the x and y coordinates are generated randomly between the minimum and the maximum values of the vehicles' source locations in a way that the edge servers are equidistant from each other. For each request submitted to an edge server, its CPU utilization, length, and size are generated randomly. The dataset containing details of these values is provided in the supplementary file and is available in a public Git repository for reproducibility.[1] Fig. 10 shows the probability distribution of the generated utilization, length, and size values for the requests.

To obtain the optimal values of EGA parameters (μ_c, μ_m, *and* pop_{size}) for the proposed algorithm, μ_c is first varied with other parameters (Table 4) being constant at their minimum values. The parameter μ_m is then varied, with μ_c constant at its optimal value and other parameters constant at their minimum values. Lastly, pop_{size} is varied keeping μ_c and μ_m constant at their optimal values and remaining parameters at their minimum values. After obtaining the optimal values for EGA parameters, the proposed algorithm is evaluated with varying latency and processing time requirements, edge–cloud bandwidth, and the number of requests. While varying each parameter, the values of the remaining three parameters are fixed to their minimum values. The performance of our proposed algorithm is measured in terms of total energy consumption for all the requests and the

[1] https://github.com/Dr-Leila-Ismail/Energy-SLA-Aware-Genetic-Algorithm-for-Edge-Cloud-Integrated-Computation-Offloading-in-Vehicular-Net.git.

Table 4
Experimental parameters.

Parameter	Value(s)
Number of vehicles	20, 40, 60, 80, 100
Requests' CPU utilization (%)	U (10,90)
Requests' length (MI)	U (500,5000)
Request's size (megabits)	U (1,16)
Vehicle — RSU bandwidth (megabits/seconds)	500
RSU — cloud bandwidth (megabits/seconds)	U (500,600), U (600,700), U (700,800), U (800,900), U (900,1000)
Requests' latency requirement (milliseconds)	100, 200, 300, 400, 500
Requests' processing time requirement (seconds)	1, 2, 3, 4, 5
Crossover rate	0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95
Mutation rate	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10
Population size	$2 \times \|M\|$, $4 \times \|M\|$, $6 \times \|M\|$, $8 \times \|M\|$, $10 \times \|M\|$
Termination condition	500 generations

U denotes Uniform distribution.

Table 5
CPU utilization and corresponding power consumption values for servers 1–6.

CPU utilization (%)	Power consumption (W)					
	Server 1	Server 2	Server 3	Server 4	Server 5	Server 6
0	138.2685	54.1	204.2420	265	45	127
10	142.2829	78.4	204.9672	531	83.7	220
20	146.7379	88.5	205.9185	624	101	254
30	151.1492	99.5	206.6314	718	118	293
40	155.3824	115	207.5923	825	133	339
50	159.9734	126	208.5179	943	145	386
60	164.4558	143	209.1885	1060	162	428
70	169.1667	165	210.2377	1158	188	463
80	173.8268	196	211.1731	1239	218	497
90	178.4852	226	211.8091	1316	248	530
100	181.7913	243	214.9755	1387	276	559

percentage of requests violating the SLAs. The percentage of SLA violations is calculated as the percentage of requests violating latency or processing time constraints as stated in using Eq. (30).

$$\%SLAVs = \left(\frac{\#_{\forall i \in M} \left([L_{r_i} \leq L_{r_i}^{max}] \vee [PT_{r_i} \leq PT_{r_i}^{max}] \right)}{|M|} \right) \times 100\% \qquad (30)$$

The proposed algorithm is compared with the following three baseline approaches to demonstrate its performance:

- ***Energy-Non-SLA-Aware Offloading using Genetic Algorithm (ENSA-GA):*** An offloading scheme using a genetic algorithm whose objective is to minimize the total energy consumption of all the requests without considering the SLA constraints.
- ***Random Offloading (RO):*** An offloading scheme where each request is randomly scheduled either at the edge server or one of the cloud serves. This scheme does not consider the servers' energy consumption and the requests' SLA requirements.
- ***No Offloading (NO):*** Each request is executed at the edge server to which it has been submitted. This scheme does not consider the servers' energy consumption and the requests' SLA requirements.

The experiments are repeated for ENSA-GA, RO, and NO with varying latency and processing time requirements, edge–cloud bandwidth, and the number of requests.

6.3. Experimental results analysis

In this section, the analyzes of the results obtained for power model development, parameters tuning for EGA, and a comparison of the proposed algorithm with ENSA-GA and RO are presented. Fig. 11 shows the CPU utilization values of our testbed (servers 1–6) stressed with a load from 0% to 100%, and the corresponding power consumptions. It also shows the developed linear regression model for each server along with its regression coefficients values. It is observed that the developed regression models fit well the actual data points, showing a linear relationship between the CPU utilization and the corresponding power consumption values.

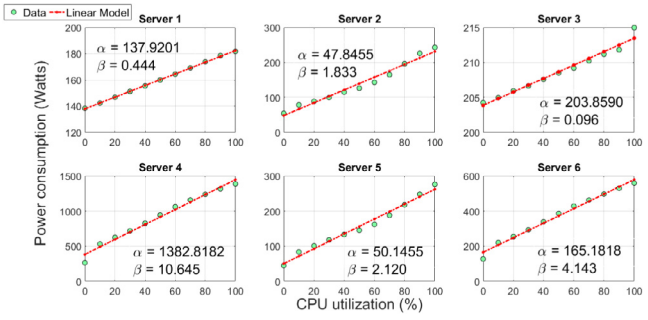
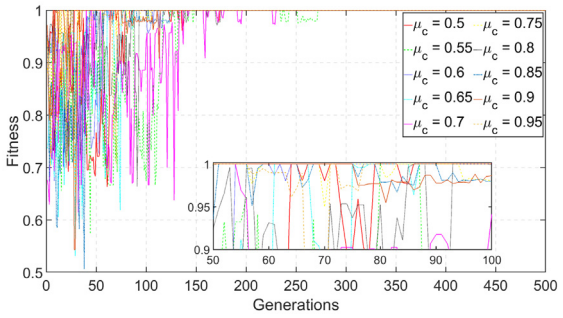
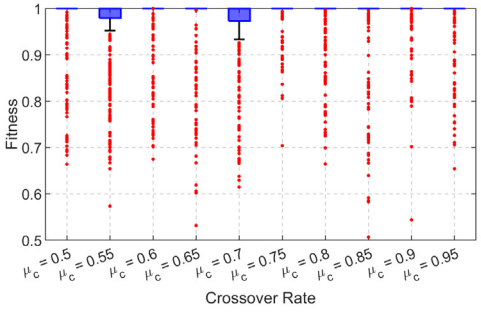


Fig. 11. Linear regression models for servers 1–6.

Fig. 12 shows the fitness score of the proposed genetic algorithm with different values of μ_c. As shown in Fig. 12(a), the fitness score for all the values of μ_c converges to 1. However, the convergence took time for $\mu_c = 0.55$ and $\mu_c = 0.7$. This is also shown in the distribution of the fitness score over generations in Fig. 12(b). As shown in Fig. 12(a), $\mu_c = 0.95$ gives the fastest convergence after the 76th generation. Figs. 13(a) and 13(b) shows the convergence of the total energy consumption and the distribution of total energy consumption, respectively, for different values of μ_c. As shown in Fig. 13, the total energy consumption for $\mu_c = 0.95$ converges quickly to the global

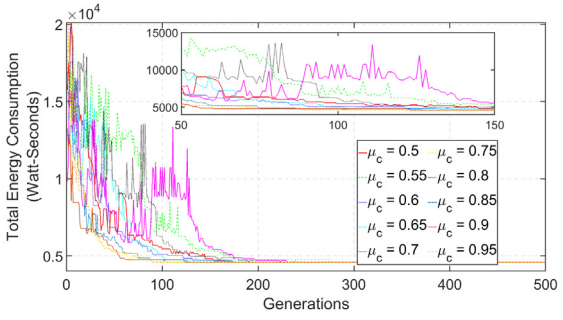


(a) Fitness score over generations

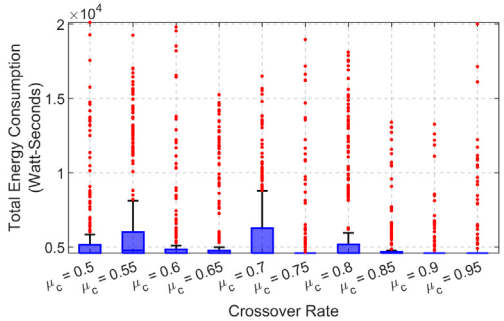


(b) Fitness score distribution over generations

Fig. 12. Fitness score of requests' offloading solutions using the proposed algorithm versus crossover rate μ_c.



(a) Total energy consumption over generations

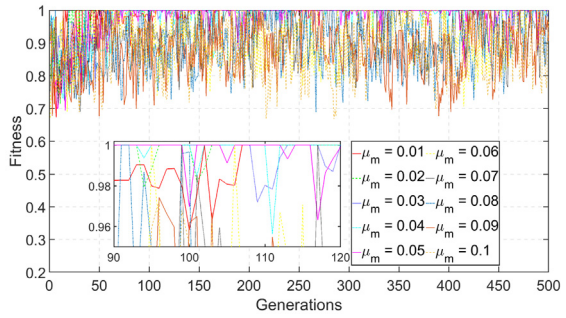


(b) Total energy consumption distribution over generations

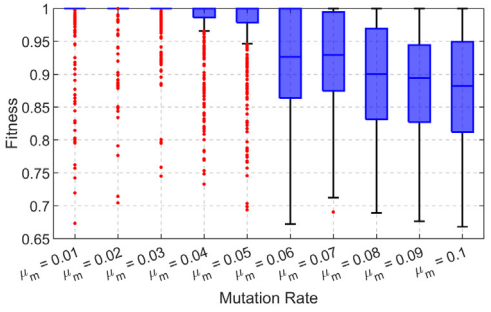
Fig. 13. Total energy consumption of requests' offloading solutions using the proposed algorithm versus crossover rate μ_c.

optima. Consequently, the optimal value of μ_c for the considered problem in this paper is set as 0.95.

Fig. 14 shows the fitness score of the proposed genetic algorithm with different values of μ_m. As shown in Fig. 14(a), the fitness score for $\mu_m = 0.06, 0.07, 0.08, 0.09, and\ 0.1$ does not converge to 1. Moreover, the score for $\mu_m = 0.04$ and $\mu_m = 0.5$ converges after 490th generation. This is also confirmed in the fitness distribution plot (Fig. 14(b)). Comparing the convergence for the remaining values of μ_m, the fastest convergence is obtained at 103rd generation for $\mu_m = 0.02$. Figs. 15(a) and 15(b) shows the convergence of the total energy consumption and the distribution of total energy consumption, respectively, for different values of μ_m. As shown in Fig. 15(a), the total energy consumption for $\mu_m = 0.02$ converges quickly to the global optima. This is also confirmed by the distribution plot in Fig. 15(b). Consequently, the optimal value of μ_m for the considered problem in this paper is set as 0.02.



(a) Fitness score over generations

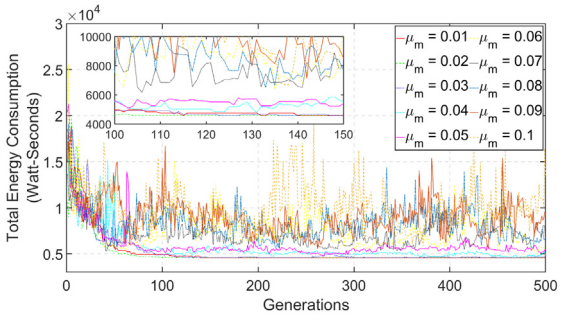


(b) Fitness score distribution over generations

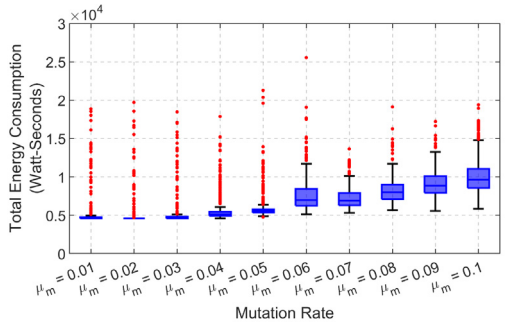
Fig. 14. Fitness score of requests' offloading solutions using the proposed algorithm versus mutation rate μ_m.

Fig. 16 shows the fitness score of the proposed genetic algorithm with different values of pop_{size}. As shown in Fig. 16(a), the fitness score for all values of pop_{size} converge to 1. This is also confirmed in the fitness distribution plot (Fig. 16(b)). However, comparing the convergence time, $pop_{size} = 8 \times |M|$ converges has the fastest convergence at 31st generation, whereas $pop_{size} = 2 \times |M|$ has the slowest convergence at 104th generation. Figs. 17(a) and 17(b) shows the convergence of the total energy consumption and the distribution of total energy consumption, respectively, for different pop_{size}. As shown in Fig. 17(a), the total energy consumption for $pop_{size} = 8 \times |M|$ converges quickly to the global optima at 49th generation, whereas that for $pop_{size} = 2 \times |M|$ has the slowest convergence at 176th generation. However, as the fitness and total energy consumption for all pop_{size} values converge before 200 generations, $pop_{size} = 2 \times |M|$ is considered in this paper. This is to increase the efficiency of the genetic algorithm. The optimal values for (μ_c, μ_m, $andpop_{size}$) are application dependent and should be determined for different applications.

Figs. 18–21 show the total energy consumption and percentage of SLA violations for the proposed, ENSA-GA, RO, and NO algorithms for different latency requirements, processing time requirement, edge–cloud bandwidths, and requests. The value of each parameter is varied while keeping others to their minimum

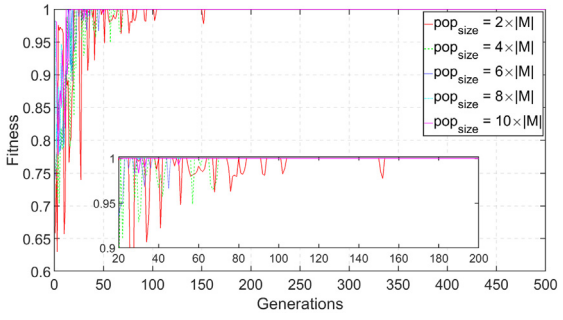


(a) Total energy consumption over generations

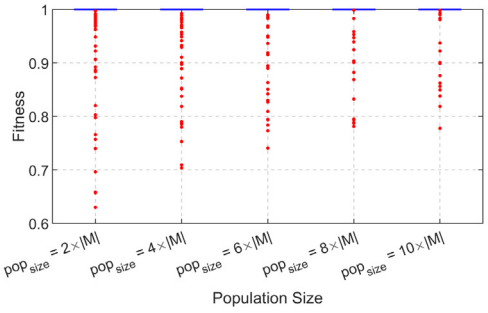


(b) Total energy consumption distribution over generations

Fig. 15. Total Energy Consumption of requests' offloading solutions using the proposed algorithm versus mutation rate μ_m.



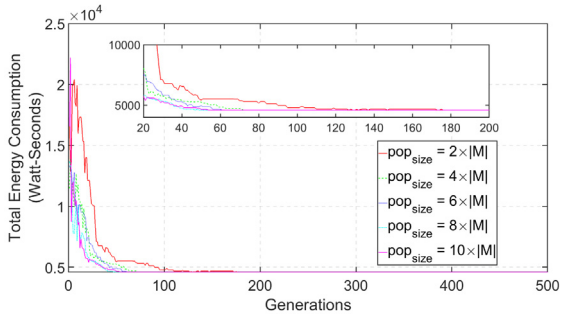
(a) Fitness score over generations



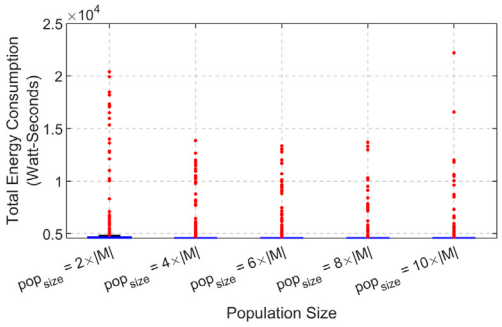
(b) Fitness score distribution over generations

Fig. 16. Fitness score of requests' offloading solutions using the proposed algorithm versus population size pop_{size}.

values (Table 4). The μ_c, μ_m, and pop_{size} parameters for the proposed and ENSA-GA algorithms are set at 0.95, 0.02, and $2 \times |M|$ respectively. As shown in Fig. 18(a), RO has the highest energy consumption, while ENSA-GA has the least energy consumption for all latency requirement values. This is because, RO does not

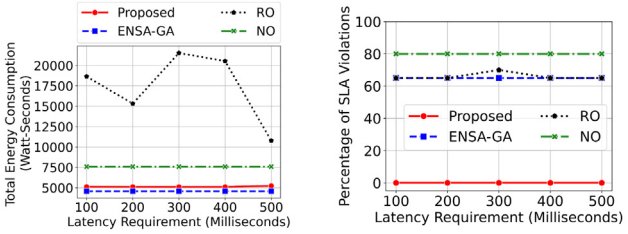


(a) Total energy consumption over generations



(b) Total energy consumption distribution over generations

Fig. 17. Total energy consumption of requests' offloading solutions using the proposed algorithm versus population size pop_{size}.

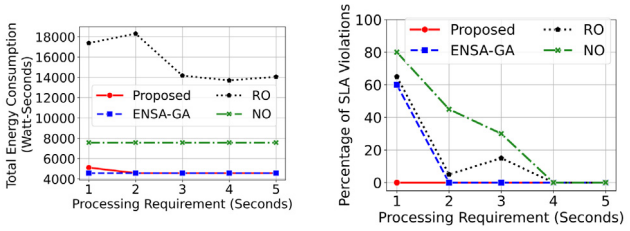


(a) Total energy consumption (b) Percentage of SLA violations

Fig. 18. Requests' offloading using the proposed algorithm, ENSA-GA, RO, and NO versus latency requirements.

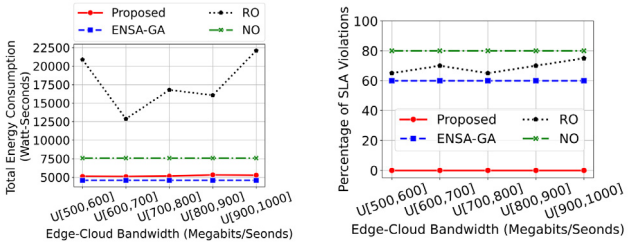
consider the energy consumption while offloading the requests, whereas the objective of ENSA-GA is to offload in a way that the total energy consumption of the requests is the minimum. The energy consumption of the proposed algorithm is higher than that of ENSA-GA because our proposed algorithm aims to minimize the total energy consumption considering the latency and processing time constraints. Consequently, the proposed algorithm will not consider an offloading scheme having the least energy consumption if any of the constraints is violated for one or more requests. Comparing the percentage of SLA violations for the considered algorithms (Fig. 18(b)), NO, RO, and ENSA-GA violate SLA constraints, whereas the proposed algorithm has no SLA violations. The average percentage of violations, with increasing latency requirements, is 65% for ENSA-GA, 66% for RO, and 80% for NO.

Figs. 19(a) and 19(b) show the total energy consumption and percentage of SLA violations respectively, for the proposed, ENSA-GA, RO, and NO algorithms for different processing time requirements. As shown in Fig. 19(a), RO has the highest energy consumption. This is because RO does not consider the energy consumption while offloading the requests. Comparing the total



(a) Total energy consumption (b) Percentage of SLA violations

Fig. 19. Requests' offloading using the proposed algorithm, ENSA-GA, RO, and NO versus processing time requirements.

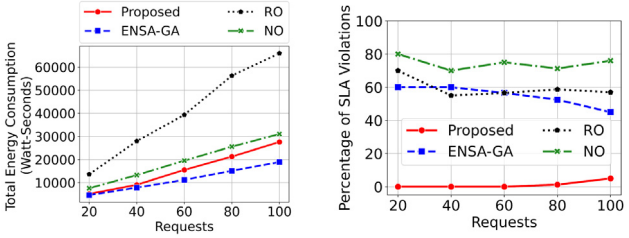


(a) Total energy consumption (b) Percentage of SLA violations

Fig. 20. Requests' offloading using the proposed algorithm, ENSA-GA, RO, and NO versus edge–cloud bandwidths.

energy consumption of the proposed algorithm and ENSA-GA, the proposed algorithm has more energy consumption for a processing time requirement of 1 s. For the requirement greater than 1 s, the total energy consumptions of the proposed and ENSA-GA algorithms are the same. This is because, for a processing requirement of 1 s, 12 requests have a processing time greater than 1 s if processed on the servers where the total energy consumption is the minimum. This is not considered by ENSA-GA. However, the proposed algorithm considers this SLA violation and offloads the requests in a way that their energy consumption is the minimum with no SLA violation. Comparing the percentage of SLA violations for the considered algorithms (Fig. 19(b)), RO and NO violate the constraints for processing time requirements of 1, 2, and 3 s, whereas ENSA-GA violates the constraints for a processing time requirement of 1 s. The proposed algorithm has no SLA violations. It is evident from the figure that the energy consumptions of the proposed and ENSA-GA algorithms are the same when ENSA-GA has no SLA violations. The average percentage of violations, with different processing time requirements, is 12% for ENSA-GA, 16% for RO, and 31% for NO.

Figs. 20(a) and 20(b) show the total energy consumption and percentage of SLA violations respectively, for the proposed, ENSA-GA, RO, and NO algorithms for different edge–cloud bandwidths. As shown in Fig. 20(a), RO has the highest energy consumption, whereas ENSA-GA has the least energy consumption. This is because RO does not consider the energy consumption while offloading the requests. Comparing the total energy consumption of the proposed algorithm and ENSA-GA, the proposed algorithm has more energy consumption. This is because our proposed algorithm aims to minimize energy consumption while considering SLA violations. Comparing the percentage of SLA violations for the considered algorithms (Fig. 20(b)), NO, RO, and ENSA-GA violate SLA constraints, whereas the proposed algorithm has no SLA violations. The average percentage of violations, with different bandwidths, is 60% for ENSA-GA, 69% for RO, and 80% for NO.



(a) Total energy consumption (b) Percentage of SLA violations

Fig. 21. Requests' offloading using the proposed algorithm, ENSA-GA, RO, and NO versus number of requests.

Figs. 21(a) and 21(b) show the total energy consumption and percentage of SLA violations respectively, for the proposed, ENSA-GA, RO, and NO algorithms for increasing requests. As shown in Fig. 21(a), RO has the highest energy consumption, whereas ENSA-GA has the least energy consumption. Comparing the total energy consumption of the proposed algorithm and ENSA-GA, the proposed algorithm has more energy consumption. The total energy consumption for all the algorithms increases with increasing requests. Comparing the percentage of SLA violations for the considered algorithms (Fig. 21(b)), NO, RO, and ENSA-GA violate SLA constraints. The number of requests violating SLA for ENSA-GA increases with an increasing number of total requests. However, the percentage of SLA violations decreases with increasing requests. This is because the percentage is calculated based on the total requests. The average percentage of violations, with increasing requests, is 54.82% for ENSA-GA, 59.47% for RO, and 60.45% for NO. The proposed algorithm has no SLA violations for 20, 40, and 60 requests. However, it violates SLA for 1.25% of requests when the total number of requests is 80, and 5% of requests when total requests are 100. This is because the algorithm is not able to converge to global optima within 500 generations for a high number of requests. The average percentage of violations, with increasing requests, for the proposed algorithm is 1.25%.

Table 6 summarizes and compares the total energy consumption and percentage of SLA violations for the proposed, ENSA-GA, RO, and NO algorithms.

7. Conclusions and future work

Computation offloading is important in edge–cloud integrated vehicular networks to execute computationally intensive applications having strict SLA requirements. However, the energy consumption of the edge–cloud integrated computing platform should be considered energy-efficiency is crucial. In this paper, an Energy-SLA-Aware evolutionary genetic algorithm is proposed for edge–cloud computation offloading in a vehicular network that executes a vehicle's request either on the edge server to which the request is submitted or offloads the request to one of the cloud servers. The offloading decision is made in a way that the total energy consumption of a set of requests is minimized and the SLA requirements of each request are maintained in terms of latency and processing time. The SLA constraints in the proposed algorithm are handled using the adaptive penalty function. This is the first work to propose an energy-SLA-aware offloading in the vehicular network using EGA that optimizes the energy consumption of the edge and cloud servers simultaneously, while adhering to the latency and processing time constraints. Comparative analysis and numerical experiments carried out revealed that the proposed algorithm outperforms no offloading and random offloading approaches in terms of energy consumption, and no offloading, random and energy-non-SLA-aware genetic-based baseline approaches in terms of

Table 6
Average total energy consumption and percentage of SLA violations for the proposed, ENSA-GA, RO, and NO algorithms.

Variable	Average total energy consumption (W-s)				Average percentage of SLA violations			
	Proposed	ENSA-GA	RO	NO	Proposed	ENSA-GA	RO	NO
Latency requirement	5135.264	4580.469	17 358.68	7584.969	0%	65%	66%	80%
Processing time requirement	4688.509	4580.469	15 526.07	7584.969	0%	12%	17%	31%
Edge–cloud bandwidth	5194.99	4580.469	17 744.25	7584.969	0%	60%	69%	80%
Requests	15 703.01	11 523.76	40 616.08	19 386.44	1.25%	54.82%	59.47%	60.45%

percentage of SLA violations. For future research work, partial offloading will be investigated where part of the request would be executed locally on the vehicle, and the remaining part(s) would be offloaded to edge and/or cloud servers while optimizing energy in vehicle/edge/cloud and preserving SLAs. In addition, different selection algorithms for the fittest offloading solutions in EGA will be evaluated. Another future work is to consider the security of the applications' requests during offloading.

CRediT authorship contribution statement

Huned Materwala: Methodology, Investigation, Writing – original draft. **Leila Ismail:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Raed M. Shubair:** Writing – review & editing. **Rajkumar Buyya:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded by the National Water and Energy Center of the United Arab Emirates University (Grant 31R215).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2022.04.009.

References

[1] S. Sharma, A. Kaul, VANETs cloud: Architecture, applications, challenges, and issues, Arch. Comput. Methods Eng. 28 (2021) 2081–2102.
[2] A. Ullah, S. Yaqoob, M. Imran, H. Ning, Emergency message dissemination schemes based on congestion avoidance in VANET and vehicular FoG computing, IEEE Access 7 (2018) 1570–1585.
[3] M. Sookhak, F.R. Yu, Y. He, H. Talebian, N.S. Safa, N. Zhao, M.K. Khan, N. Kumar, Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing, IEEE Veh. Technol. Mag. 12 (3) (2017) 55–64.
[4] K. Mershad, O. Cheikhrouhou, L. Ismail, Proof of accumulated trust: A new consensus protocol for the security of the IoV, Veh. Commun. 32 (2021) 100392.
[5] P. Mell, T. Grance, et al., The NIST definition of cloud computing, 2011.
[6] L. Ismail, L. Zhang, Information Innovation Technology in Smart Cities, Springer, 2018.
[7] S. Raza, S. Wang, M. Ahmed, M.R. Anwar, A survey on vehicular edge computing: architecture, applications, technical issues, and future directions, Wirel. Commun. Mob. Comput. 2019 (2019).
[8] W.Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: A survey, Future Gener. Comput. Syst. 97 (2019) 219–235.
[9] L. Ismail, H. Materwala, IoT-edge-cloud computing framework for qos-aware computation offloading in autonomous mobile agents: Modeling and simulation, in: International Conference on Mobile, Secure, and Programmable Networking, Springer, 2020, pp. 161–176.
[10] L. Ismail, H. Materwala, EATSVM: energy-aware task scheduling on cloud virtual machines, Procedia Comput. Sci. 135 (2018) 248–258.
[11] L. Ismail, A.A. Fardoun, Energy-aware task scheduling (EATS) framework for efficient energy in smart cities cloud computing infrastructures, Int. J. Therm. Environ. Eng. 13 (1) (2016) 37–48.
[12] L. Ismail, H. Materwala, Machine learning-based energy-aware offloading in edge-cloud vehicular networks, Procedia Comput. Sci. 191 (2021) 328–336.
[13] L. Ismail, B. Mills, A. Hennebelle, A formal model of dynamic resource allocation in grid computing environment, in: 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, IEEE, 2008, pp. 685–693.
[14] R. Mahmud, S.N. Srirama, K. Ramamohanarao, R. Buyya, Quality of experience (QoE)-aware placement of applications in fog computing environments, J. Parallel Distrib. Comput. 132 (2019) 190–203.
[15] L. Ismail, Dynamic resource allocation mechanisms for grid computing environment, in: 2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, IEEE, 2007, pp. 1–5.
[16] S. Yangui, A. Goscinski, K. Drira, Z. Tari, D. Benslimane, Future generation of service-oriented computing systems, Future Gener. Comput. Syst. 118 (2021) 252–256, http://dx.doi.org/10.1016/j.future.2021.01.019, URL https://www.sciencedirect.com/science/article/pii/S0167739X21000297.
[17] Y. Liu, X. Wei, J. Xiao, Z. Liu, Y. Xu, Y. Tian, Energy consumption and emission mitigation prediction based on data center traffic and PUE for global data centers, Global Energy Interconnect. 3 (3) (2020) 272–282.
[18] L. Belkhir, A. Elmeligi, Assessing ICT global emissions footprint: Trends to 2040 & recommendations, J. Cleaner Prod. 177 (2018) 448–463.
[19] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, P. Zhang, Energy-aware mobile edge computation offloading for IoT over heterogenous networks, IEEE Access 7 (2019) 13092–13105.
[20] H. Guo, J. Zhang, J. Liu, H. Zhang, Energy-aware computation offloading and transmit power allocation in ultradense IoT networks, IEEE Internet Things J. 6 (3) (2018) 4317–4329.
[21] Z. Ning, J. Huang, X. Wang, J.J. Rodrigues, L. Guo, Mobile edge computing-enabled internet of vehicles: Toward energy-efficient scheduling, IEEE Netw. 33 (5) (2019) 198–205.
[22] X. Xu, Y. Li, T. Huang, Y. Xue, K. Peng, L. Qi, W. Dou, An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks, J. Netw. Comput. Appl. 133 (2019) 75–85.
[23] Y. Zhai, W. Sun, J. Wu, L. Zhu, J. Shen, X. Du, M. Guizani, An energy aware offloading scheme for interdependent applications in software-defined IoV with fog computing architecture, IEEE Trans. Intell. Transp. Syst. 22 (6) (2020) 3813–3823.
[24] J. Zhang, X. Hu, Z. Ning, E.C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Internet Things J. 5 (4) (2017) 2633–2645.
[25] Z. Li, V. Chang, J. Ge, L. Pan, H. Hu, B. Huang, Energy-aware task offloading with deadline constraint in mobile edge computing, EURASIP J. Wireless Commun. Networking 2021 (1) (2021) 1–24.
[26] X. Huang, L. He, W. Zhang, Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network, in: 2020 IEEE International Conference on Edge Computing (EDGE), IEEE, 2020, pp. 1–8.
[27] X. Huang, K. Xu, C. Lai, Q. Chen, J. Zhang, Energy-efficient offloading decision-making for mobile edge computing in vehicular networks, EURASIP J. Wireless Commun. Networking 2020 (1) (2020) 1–16.
[28] L. Pu, X. Chen, G. Mao, Q. Xie, J. Xu, Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications, IEEE Internet Things J. 6 (1) (2018) 84–99.
[29] M. Goudarzi, H. Wu, M. Palaniswami, R. Buyya, An application placement technique for concurrent IoT applications in edge and fog computing environments, IEEE Trans. Mob. Comput. 20 (4) (2020) 1298–1311.
[30] K. Peng, M. Zhu, Y. Zhang, L. Liu, J. Zhang, V.C. Leung, L. Zheng, An energy- and cost-aware computation offloading method for workflow applications in mobile edge computing, EURASIP J. Wireless Commun. Networking 2019 (1) (2019) 1–15.
[31] L. Ismail, H. Materwala, ESCOVE: Energy-SLA-aware edge–cloud computation offloading in vehicular networks, Sensors 21 (15) (2021) 5233.

[32] H. Materwala, L. Ismail, Performance and energy-aware bi-objective tasks scheduling for cloud data centers, Procedia Comput. Sci. 197 (2022) 238–246.
[33] R. Zhang, F. Tian, X. Ren, Y. Chen, K. Chao, R. Zhao, B. Dong, W. Wang, Associate multi-task scheduling algorithm based on self-adaptive inertia weight particle swarm optimization with disruption operator and chaos operator in cloud environment, Serv. Orient. Comput. Appl. 12 (2) (2018) 87–94.
[34] X. Fang, W. Wang, L. He, Z. Huang, Y. Liu, L. Zhang, Research on improved NSGA-II algorithm and its application in emergency management, Math. Probl. Eng. 2018 (2018).
[35] H. Hao, C. Xu, L. Zhong, G.-M. Muntean, A multi-update deep reinforcement learning algorithm for edge computing service offloading, in: Proceedings of the 28th ACM International Conference on Multimedia, 2020, pp. 3256–3264.
[36] M. Wiering, Memory-based memetic algorithms, in: Benelearn'04: Proceedings of the Thirteenth Belgian-Dutch Conference on Machine Learning, 2004, pp. 191–198.
[37] D.K. Mishra, V. Shinde, 6 A review of global optimization problems using meta-heuristic algorithm, in: A. Khamparia, A. Khanna, N.G. Nguyen, B.L. Nguyen (Eds.), Nature-Inspired Optimization Algorithms: Recent Advances in Natural Computing and Biomedical Applications, De Gruyter, 2021, pp. 87–106, http://dx.doi.org/10.1515/9783110676112-006.
[38] L. Ismail, E.H. Abed, Linear power modeling for cloud data centers: taxonomy, locally corrected linear regression, simulation framework and evaluation, IEEE Access 7 (2019) 175003–175019.
[39] L. Ismail, H. Materwala, Computing server power modeling in a data center: Survey, taxonomy, and performance evaluation, ACM Comput. Surv. 53 (3) (2020) 1–34.
[40] M. Thakur, S.S. Meghwani, H. Jalota, A modified real coded genetic algorithm for constrained optimization, Appl. Math. Comput. 235 (2014) 292–317.
[41] L. Liu, M. Zhang, R. Buyya, Q. Fan, Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing, Concurr. Comput.: Pract. Exper. 29 (5) (2017) e3942.
[42] M. Akbari, H. Rashidi, S.H. Alizadeh, An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems, Eng. Appl. Artif. Intell. 61 (2017) 35–46.
[43] SPECpower, SPECpower benchmark, 2009, https://www.spec.org/power_ssj2008/, (Accessed on 09/28/2021).
[44] SPECpower, Server 2: SPECpower_ssj2008, 2021, https://www.spec.org/power_ssj2008/results/res2012q1/power_ssj2008-20120306-00434.html, (Accessed on 09/28/2021).
[45] SPECpower, Server 4: SPECpower_ssj2008, 2021, https://www.spec.org/power_ssj2008/results/res2012q1/power_ssj2008-20120306-00437.html, (Accessed on 09/28/2021).
[46] SPECpower, Server 5: SPECpower_ssj2008, 2021, https://www.spec.org/power_ssj2008/results/res2016q1/power_ssj2008-20151215-00708.html, (Accessed on 09/28/2021).
[47] SPECpower, Server 6: SPECpower_ssj2008, 2021, https://www.spec.org/power_ssj2008/results/res2012q1/power_ssj2008-20120213-00420.html, (Accessed on 09/28/2021).
[48] G. Carlucci, CPULoadGenerator, 2017, https://github.com/GaetanoCarlucci/CPULoadGenerator, (Accessed on 09/28/2021).
[49] Tektronix, TBS 2000 digital oscilloscope, 2021, https://www.tek.com/oscilloscope/tbs2000-basic-oscilloscope, (Accessed on 09/28/2021).
[50] D. Yang, L. Li, K. Redmill, U. Özgüner, Top-view trajectories: A pedestrian dataset of vehicle-crowd interaction from controlled experiments and crowded campus, in: 2019 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2019, pp. 899–904.
[51] N. Auluck, A. Azim, K. Fizza, Improving the schedulability of real-time tasks using fog computing, IEEE Trans. Serv. Comput. (2019).
[52] A. Jaddoa, G. Sakellari, E. Panaousis, G. Loukas, P.G. Sarigiannidis, Dynamic decision support for resource offloading in heterogeneous Internet of Things environments, Simul. Model. Pract. Theory 101 (2020) 102019.
[53] J. Almutairi, M. Aldossary, A novel approach for IoT tasks offloading in edge-cloud environments, J. Cloud Comput. 10 (1) (2021) 1–19.

Huned Materwala is currently working towards his Ph.D. degree at the Intelligent Distributed Computing and Systems (INDUCE) Research Laboratory at the College of Information Technology of the United Arab Emirates University (UAEU), United Arab Emirates. He has been working as a Research Assistant at INDUCE Lab and been awarded a Ph.D. scholarship from UAEU to support his studies. His research interests include Edge–Cloud Computing, Internet of Things (IoT), and Distributed Systems.

Dr. Leila Ismail is the Founder and Director of the Intelligent Clouds and Distributed Computing Systems (INDUCE) Research Laboratory at the College of Information technology of the United Arab Emirates University (UAEU), and an Associate Professor at the Department of Computer Science and Software Engineering. She has vast industrial and academic experience at Sun Microsystems Research & Development Center in France, working the design and implementation of highly available distributed systems, and participated in the deposit of a US patent. She served in teaching at Grenoble I, France, and has been serving as an Adjunct Professor at the Digital Ecosystems and Business Intelligence Institute Curtin University, Australia. She has been very active in creating smart and efficient digital ecosystems responding to nowadays emergency needs for better living in our dynamic global habitat, introducing blockchain, Internet of Things (IoT), machine learning, deep learning, and Artificial Intelligence approaches for different applications domains, such as healthcare, energy, and smart transportation. She has been very active in international research collaborations within, Australia, and USA, and been invited as a keynote speaker in several conferences, including Women in Data Science International Conference (WiDS 2021), organized by Stanford University. She is the recipient of several awards and appreciation certificates, including the IBM Shared University Research (SURA) and the IBM Faculty Awards, very competitive worldwide, the UAE University Award for high achievements publishing in top ranked journals. She won funding for major projects as Principal Investigator, on grid and cloud computing, intelligent systems and smart applications, and awards of top achievements. She served as Associate Editor of the International Journal of Parallel, Emergent, and Distributed Systems for several years. She has been participating in the success of many IEEE and ACM international conferences in several roles, such as General Chair, Organizing Committee Chair, and Technical Program Chair. She is the author of many scientific publications in journals and conferences, and the Editor of the Information Innovation Technology in Smart Cities book, published by Springer Nature.

Dr. Raed M. Shubair (Senior Member, IEEE) received the B.Sc. (with Distinction and First Class Hons.) degree in electrical engineering from Kuwait University, Kuwait, in June 1989, and the Ph.D. (with Distinction) degree in electrical engineering from the University of Waterloo, Canada, in February 1993, for which he received the University of Waterloo Distinguished Doctorate Dissertation Award. He is a Senior Advisor in the Office of Undersecretary for Academic Affairs, Ministry of Education, UAE. He is also a Full Professor of Electrical Engineering and Chair of IEEE at New York University (NYU) Abu Dhabi. His current and past academic and research appointments also include Massachusetts Institute of Technology (MIT), Harvard University, and University of Waterloo. He has been a Full Professor of Electrical Engineering with Khalifa University (formerly, Etisalat University College), UAE, which he joined in 1993 up to 2017, during which he received several times the Excellence in Teaching Award and Distinguished Service Award. He has over 380 publications in the form of articles in peer-reviewed journals, papers in referred conference proceedings, book chapters, and US patents. His publication span several research areas, including 6G and terahertz communications, modern antennas and applied electromagnetics, signal and array processing, machine learning, IoT and sensor localization, medical sensing and nano-biomedicine. He is recipient of several international awards, including the Distinguished Service Award from ACES Society, USA and from MIT Electromagnetics Academy, USA. He organized and chaired numerous technical special sessions and tutorials in IEEE flagship conferences. He delivered more than 60 invited speaker seminars and technical talks in world-class universities and flagship conferences. He served as an invited speaker with the U.S. National Academies of Sciences, Engineering, and Medicine Frontiers Symposium. He is a standing member of the editorial boards of several international journals and serves regularly on the steering, organizing, and technical committees of IEEE flagship conferences in Antennas, Communications, and Signal Processing, including several editions of IEEE AP-S/URSI, EuCAP, IEEE GloablSIP, IEEE WCNC, and IEEE ICASSP. He has served as the TPC Chair of IEEE MMS2016 and TPC Chair of IEEE GlobalSIP 2018 Symposium on 5G Satellite Networks. He holds several leading roles in the international professional engineering community. He holds several leading roles in the international professional engineering community. He is a Board Member of the European School of Antennas, Regional Director for the IEEE Signal Processing Society in IEEE Region 8 Middle East and served as the founding chair of the IEEE Antennas and Propagation Society Educational Initiatives Program. He is a Fellow of MIT Electromagnetics Academy and a Founding Member of MIT Scholars of the Emirates. He is Editor for the IEEE Journal of Electromagnetics, RF, and Microwaves in Medicine and Biology, and Editor for the IEEE Open Journal of Antennas and Propagation. He is a Founding Member of five IEEE society chapters in UAE, which are IEEE Communication Society Chapter, IEEE Signal

Processing Society Chapter, IEEE Antennas and Propagation Society Chapter, IEEE Microwave Theory and Techniques Society Chapter, and IEEE Engineering in Medicine and Biology Society Chapter. He is the Founder and Chair of IEEE at New York University Abu Dhabi. He is an officer for IEEE ComSoc emerging technical initiative (ETI) on Machine Learning for Communications. He is the founding director of IEEE UAE Distinguished Seminar Series Program for which he was selected to receive, along with Mohamed AlHajri of MIT, the 2020 IEEE UAE Award of the Year.

Dr. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 850 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. Dr. Buyya is one of the highly cited authors in computer science and software engineering worldwide (h-index=150, g-index=322, and 117,200+ citations). Dr. Buyya is recognized as Web of Science "Highly Cited Researcher" for five consecutive years since 2016, IEEE Fellow, Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier, and the "Best of the World", in Computing Systems field, by The Australian 2019 Research Review. Software technologies for Grid, Cloud, and Fog computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 50+ countries around the world. Manjrasoft's Aneka Cloud technology developed under his leadership has received "Frost New Product Innovation Award". He served as founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Editor-in-Chief of Software: Practice and Experience, a long standing journal in the field established 50+ years ago.