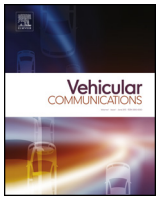
Vehicular Communications 43 (2023) 100654

Contents lists available at ScienceDirect

Vehicular Communications

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



QoS-SLA-aware adaptive genetic algorithm for multi-request offloading in integrated edge-cloud computing in Internet of vehicles

Huned Materwala [a,b], Leila Ismail [a,b,c,*], Hossam S. Hassanein [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, United Arab Emirates
[b] National Water and Energy Center, United Arab Emirates University, United Arab Emirates
[c] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia
[d] School of Computing, Queen's University, Canada

ARTICLE INFO

Article history:
Received 11 November 2022
Received in revised form 3 July 2023
Accepted 26 July 2023
Available online 1 August 2023

Keywords:
Artificial intelligence (AI)
Computation offloading
Genetic algorithm (GA)
Intelligent transportation system
Internet of things (IoT)-edge-cloud computing
Internet of vehicles (IoV)

ABSTRACT

The Internet of Vehicles over vehicular ad hoc network is an emerging technology enabling the development of smart applications focused on improving traffic safety, traffic efficiency, and the overall driving experience. These applications have stringent requirements detailed in the Service Level Agreement. Since vehicles have limited computational and storage capabilities, applications' requests are offloaded onto an integrated edge-cloud computing system. Existing offloading solutions focus on optimizing the application's Quality of Service (QoS) in terms of execution time while respecting a single SLA constraint. They do not consider the impact of overlapped multi-request processing nor the vehicle's varying speed. This paper proposes a novel Artificial Intelligence QoS-SLA-aware adaptive genetic algorithm (QoS-SLA-AGA) to optimize the application's execution time for multi-request offloading in a heterogeneous edge-cloud computing system, which considers the impact of processing multi-requests overlapping and dynamic vehicle speed. The proposed genetic algorithm integrates an adaptive penalty function to assimilate the SLA constraints regarding latency, processing time, deadline, CPU, and memory requirements. Numerical experiments and analysis compare our QoS-SLA-AGA to baseline genetic-based, meta-heuristic Particle Swarm Optimization (PSO), random offloading, All Edge Computing (AEC), and All Cloud Computing (ACC) approaches. Results show QoS-SLA-AGA executes the requests 1.04, 1.23, 1.05, and 9.41 times faster on average compared to the PSO, random offloading, ACC, and AEC approaches respectively. Moreover, the proposed algorithm violates 49.58%, 60.36%, 16.26%, and 80.42% fewer SLAs compared to PSO, random, ACC, and AEC respectively. In contrast, the baseline genetic-based approach increases the requests' performance by 1.14 times, with 24.03% more SLA violations.

1. Introduction

Internet of Vehicles (IoV) over Vehicular Ad-hoc Networks (VANETS) is a self-organizing network of vehicles equipped to exchange data between mobile vehicles and infrastructure [1]. The vehicles act as smart nodes having sensing, computing, storage, and networking capabilities [2], [3]. Data exchange is realized using vehicle-to-vehicle (V2V), vehicle-to-roadside (V2R), vehicle-to-infrastructure (V2I), vehicle-to-cloud (V2C), and vehicle-to-pedestrian (V2P) communication. IoV provides mechanisms to develop applications for safe driving and efficient traffic management [4]; such applications include accident prevention, infotainment, real-time navigation, image processing, and pattern recognition for autonomous driving. However, the vehicle's limited computation and storage capabilities hinder the deployment of these compute-intensive and time-critical applications.

Vehicular cloud computing (VCC) [5] has been developed to enable compute-intensive vehicles' requests to be processed on remote cloud servers [6] to comply with the processing and resource requirements of Service Level Agreements (SLAs). However, latency require-

* 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, United Arab Emirates.
E-mail address: leila@uaeu.ac.ae (L. Ismail).

https://doi.org/10.1016/j.vehcom.2023.100654

ments of communication-bound applications may be violated due to long-distance data transmission between vehicles and remote cloud servers.

Consequently, Vehicular Edge Computing (VEC) [7] pushed cloud services to the edge of the radio access network in closer proximity to the mobile vehicles, thus reducing communication delay. However, the VEC servers (deployed within Roadside Units (RSUs)) may violate the stringent deadline constraints of compute-intensive applications due to their limited computing capabilities. Consequently, it becomes necessary to develop a mechanism to offload vehicular requests onto an integrated edge-cloud computing system to comply with SLA requirements of latency for communication-bound applications (e.g., traffic alert and accident prevention) and processing for computation-bound applications (e.g., computer vision and multimedia), while optimizing applications' Quality of Service (QoS) [8].

Several works proposed computation offloading algorithms for an integrated edge-cloud computing system in the Internet of Things (IoT)/IoV networks. They focus on optimizing the applications' QoS without considering SLA requirements [9–15], or respecting SLA without QoS optimization [16–21]. Some works proposed QoS-SLA-aware offloading solutions [22–29]. To the best of our knowledge, no work has considered the impact of multi-requests overlapping in heterogeneous edge-cloud computing system servers. Thus, we propose a novel Artificial Intelligence QoS-SLA-aware adaptive genetic algorithm (QoS-SLA-AGA) for offloading vehicular requests. It aims to optimize the QoS by minimizing the total execution time of the vehicular requests while respecting SLAs in terms of latency (the total communication time of the request), processing time, deadline (the summation of communication and computation times), CPU, and memory requirements via an adaptive penalty function. The algorithm learns from selected search space solutions to find an improved one. In addition, the proposed offloading algorithm considers the impact of executing multiple requests in edge/cloud resources on application performance in IoV. The specific contributions of this work in the field of computational offloading in the IoV are the following:

- We formulate an optimization algorithm for multi-request offloading in a heterogeneous integrated edge-cloud computing system for IoV that minimizes the total execution time of vehicular requests while respecting the requests' SLA requirements.
- We propose a novel QoS-SLA-AGA to solve the formulated constrained optimization problem via an adaptive penalty function.
- We provide complexity and convergence analysis of the proposed algorithm.
- We conduct experiments to obtain the optimal values of GA's parameters driving the algorithm towards convergence.
- We compare the performance of the proposed algorithm with baseline genetic-based, meta-heuristic Particle Swarm Optimization (PSO), random offloading, all edge computing, and all cloud computing approaches in terms of total execution time (seconds) and SLA violations (SLAVs) with varying SLA requirements or the number of requests.

The remainder of this article is organized as follows. In Section 2, we discuss related work. Section 3 describes the system model of an integrated edge-cloud computing system for the IoV. We formulate the offloading optimization problem in Section 4. Section 5 explains our proposed QoS-SLA-AGA algorithm for offloading. Section 6 presents our numerical experiments, comparative analysis, and performance results. Finally, we conclude and suggest future directions in Section 7.

2. Related work

We divide the current literature on offloading for an IoT-edge-cloud integrated computing system into three categories:

1) QoS-aware offloading that optimizes applications' QoS without considering SLA requirements [9–15],
2) SLA-aware offloading that respects applications' SLA constraints without enhancing the QoS [16–21], and
3) QoS-SLA-aware offloading that optimizes applications' QoS while respecting SLA constraints [22–29].

QoS-aware offloading solutions, Pham et al. [9] proposed a game-theoretic approach to execute mobile device requests locally on the device or an edge server. The algorithm optimizes the weighted sum of the request's execution time and the device's energy consumption. However, the reply's communication time and the device's mobility are not considered. Xu et al. [10] proposed a decomposition-based evolutionary algorithm to offload vehicular requests on edge nodes such that the total execution time of the request is minimized, and the resource utilization of the edge nodes is maximized. However, the mobility of the vehicle outside the communication range of the edge executing the request is not considered. Similarly, the deep learning [11], [12], the Non-dominated Sorting GA (NSGA) III [13], and the Particle Swarm Optimization [14], [15] offloading algorithms do not consider the mobility of the vehicles/devices when optimizing the requests' execution times. In addition, algorithms in [9–14] do not consider edge/cloud servers' heterogeneity.

SLA-aware offloading solutions, the optimization algorithms in [16–20] schedule mobile devices' requests to either edge, edge/cloud, or mobile/edge to optimize requests' acceptance, edge/cloud profit, mobile devices and edges servers energy, requests acceptance along with edge operational cost, and network usage respectively. The algorithms used in these works are based on Lagrangian relaxation, simulated annealing, deep reinforcement learning, or heuristic. Wu et al. [21] proposed an offloading algorithm either locally on the IoT device or edge/cloud such that the total energy consumption of the IoT device is the minimum. These works consider a request's total execution time as a constraint. However, the algorithms [16–21] do not consider the devices' mobility and heterogeneity among the edge/cloud servers. Moreover, the communication time to deliver the request's reply is not considered in [18], [21].

QoS-SLA-aware offloading solutions, the algorithms in [22], [23] offload IoT devices' requests to edge/cloud using dynamic switching and fuzzy logic respectively. The requests' execution times are optimized in [22] and the execution time and edges' resource utilization in [23]. These algorithms consider the requests' execution times as constraints. Algorithms in [24–27] schedule vehicles' requests either on the vehicle, vehicle/edge, or vehicle/edge/cloud. The total execution time of the requests and the computational costs are optimized in [24], [25] using game theory, whereas the total execution time and load balancing on the edges are optimized using mixed-integer non-linear programming in [26]. Zhu et al. [27] minimize the weighted sum of maximum execution time and total quality loss using linear programming and particle swarm optimization. A request's execution time is considered a constraint in [24–26], whereas the execution time and the vehicle's available CPU and memory are considered constraints in [27]. Peng et al. [28] proposed NSGA II and strength Pareto evolutionary algorithm to schedule mobile device's requests either locally or edge/cloud such that summation of the total execution time of the requests and the device's energy consumption is minimized while respecting the constraint on the request's execution time. Liao

Table 1
Summary of past works on QoS, SLA, and QoS-SLA-aware computation offloading in an integrated IoT-edge-cloud computing system.

Work	Algorithm	Considered system components for requests processing			Optimization objective function	Considered requests' SLA requirements					Edge servers' heterogeneity consideration	Cloud servers' heterogeneity consideration	Request-reply delivery time consideration	Multi-request consideration	IoT/Vehicle mobility consideration	Dynamic vehicle speed consideration
		IoT/Vehicle	Edge	Cloud		Latency	Processing time	Deadline	CPU requirement	Memory requirement						
QoS-Aware																
[9]	Game theory	✓	✓	✗	Minimize the weighted sum of energy consumption and execution time	✗	✗	✗	✗	✗	✗	NA	✗	✗	✗	NA
[10]	Decomposition-based evolutionary algorithm	✗	✓	✗	Minimize total execution time and maximize resource utilization	✗	✗	✗	✗	✗	✗	NA	✓	✗	✗	NA
[11]	Deep learning	✓	✓	✓	Minimize the weighted sum of total execution time and vehicles' energy consumption	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	NA
[12]	Q function of deep neural network	✓	✓	✗	Maximize resource utilization while minimizing end-to-end delay	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	NA
[13]	NSGA III	✓	✓	✓	Minimize total execution time and mobile devices' energy consumption	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	NA
[14]	Genetic algorithm and PSO	✗	✓	✓	Minimize task response time	✗	✗	✗	✗	✗	NR	NR	✓	✗	✗	NA
[15]	PSO	✗	✓	✓	Minimize the weighted sum of latency and mobile devices' energy consumption	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	NA
SLA-Aware																
[16]	Lagrangian relaxation	✗	✓	✗	Maximize the weighted sum of admitted requests for processing	✗	✗	✓	✗	✗	NR	NA	✓	✗	✓	✗
[17]	Simulated annealing	✗	✓	✓	Maximize cost profit of edge/cloud systems	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	NA
[18]	Deep reinforcement learning	✓	✓	✗	Minimize energy consumption of mobile devices and edge servers	✗	✗	✓	✗	✗	✗	NA	✗	✗	✗	NA
[19]	NR	✗	✓	✗	Maximize requests' admissions and minimize admission cost	✗	✗	✓	✗	✗	✗	NA	✓	✗	✗	NA
[20]	Heuristic	✗	✓	✓	Minimize network resource usage and maximize requests' admissions	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	NA
[21]	Lyapunov optimization	✓	✓	✓	Minimize energy consumption of IoT devices	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	NA
QoS-SLA-Aware																
[22]	Dynamic switching	✗	✓	✓	Minimize total execution time	✗	✗	✓	✗	✗	✗**	✗	✗	✗	✗	NA
[23]	Fuzzy logic	✗	✓	✓	Minimize total execution time and maximize resource utilization	✗	✗	✓	✗	✗	✓	✗	✓	✗	✗	NA
[24]	Game theory	✓	✓	✗	Minimize total execution time and offloading cost	✗	✗	✓	✗	✗	✗	NA	✓	✗	✗	NA
[25]	Game theory and Lagrange multiplier	✓	✓	✓	Minimize the weighted sum of execution time and cost of computational resource	✗	✗	✓	✗	✗	✗	✗	✗*	✗	✓	✓
[26]	Mixed integer non-linear programming	✓	✓	✗	Minimize system utility in terms of execution time and load balancing	✗	✗	✓	✗	✗	✓	NA	✗	✗	✓	✗
[27]	Linear programming and PSO	✓	✗	✗	Minimize the weighted sum of maximum execution time and total loss quality	✗	✗	✓	✓	✗	NA	NA	✗	✗	✗	NA
[28]	NSGA II and SPEA	✓	✓	✓	Minimize total execution time and mobile devices' energy consumption	✗	✗	✓	✗	✗	NR	NR	✗	✗	✗	NA
[29]	Upper confidence bound algorithm	✓	✓	✗	Minimize deadline while maximizing throughput	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗	NA
This paper	Adaptive genetic algorithm	✗	✓	✓	Minimize total execution time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓– Considered; ✗ – Not considered; NA – Not Applicable; NR – Not Reported; NSGA – Non-dominated Sorting Genetic Algorithm; PSO – Particle Swarm Optimization; SPEA – Strength Pareto Evolutionary Algorithm.

* Considered for cloud processing and not for edge processing.

** Heterogeneous in terms of disk size and not processing capabilities.

et al. [29] proposed an intent-aware upper confidence bound algorithm to maximize throughput and minimize deadline while respecting long-term latency constraints. However, the algorithms in [23], [24], [27–29] do not consider the mobility of the IoT devices/vehicles, and [22], [25], [27–29] do not consider communication time to deliver the reply of the request. The heterogeneity in the edge and cloud servers is not considered by [24], [25].

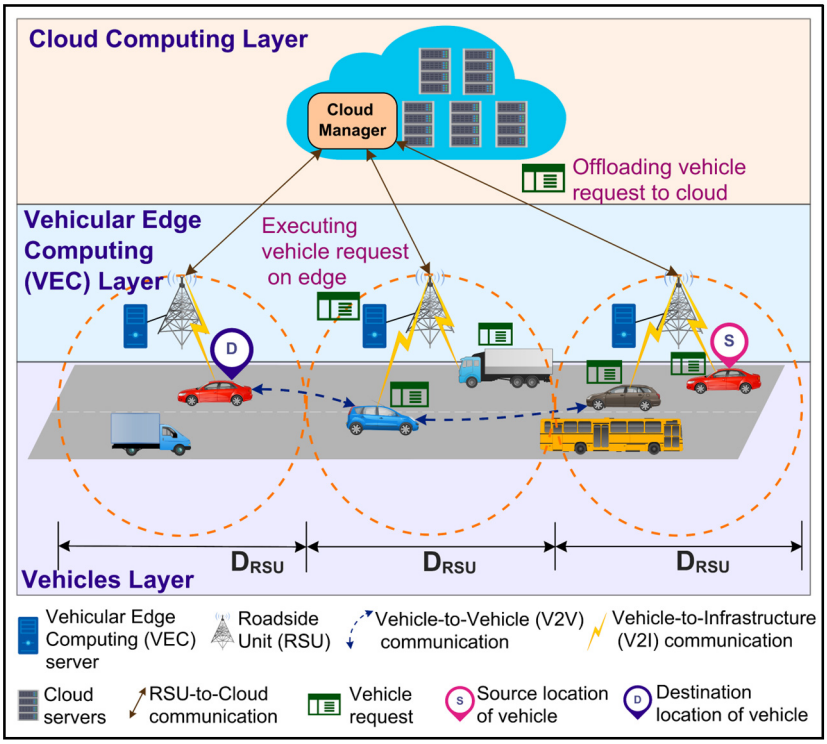


Fig. 1. Integrated edge-cloud computing system model for the Internet of Vehicles.

Table 1 presents the summary of past works on computation offloading in an integrated IoT-edge-cloud computing system. For each work, it highlights the used algorithms, considered system components for processing the requests, the objective function used for optimization, and the SLA requirements. In addition, the table shows whether those works consider the heterogeneity among edge and cloud servers, communication time to transmit the reply to the IoT device/vehicle, requests overlapping, mobility of IoT devices/vehicles, and dynamic speed of the device/vehicle. As shown in the table, among works on QoS-SLA-aware offloading [22–29], only [25] considers the dynamic speed of the vehicle and [26] considers memory requirements as SLA. To the best of our knowledge, no work considers the impact of multi-requests overlapping on the offloading decision and the dynamic speed of vehicles. In this paper, we propose a QoS-SLA-AGA for offloading in an integrated edge-cloud computing system for IoV that aims to improve the QoS by minimizing the total execution time of the applications' requests while respecting the SLAs in terms of latency, processing time, deadline, CPU, and memory requirements. Furthermore, our algorithm considers the impact of executing multiple requests in edge/cloud resources on application performance in IoV.

3. System model

Fig. 1 shows our integrated edge-cloud computing system model for vehicular networks that consists of three layers: 1) vehicles, 2) VEC, and 3) cloud computing. The first layer consists of H vehicles moving with dynamic speed on a bi-directional road. Each vehicle v_h $(h \in \mathcal{H})$ travels from a source to the destination location and has an application request r_i $(i \in \mathcal{I})$ that should be executed. A request is represented as a tuple $r_i \triangleq \left(\psi_{r_i}, \sigma_{r_i}, \varphi_{r_i}, L_{r_i}^{max}, P_{r_i}^{max}, D_{r_i}^{max}, S_{v_h}(t), \left(x_{v_h,r_i}^{src}, y_{v_h,r_i}^{src}\right), \left(x_{v_h,r_i}^{des}, y_{v_h,r_i}^{des}\right)\right)$. Requests in our system model are atomic and cannot be further divided into sub-requests. Consequently, each request can be executed on at most one edge/cloud server. The requests vary in terms of computational requirement (i.e., length, CPU, and memory utilization values) and communication demand (i.e., data size).

The second layer (i.e., VEC) consists of J RSUs placed alongside the road at equidistant. Each RSU_j $(j \in \mathcal{J})$ has a coverage range of D_{RSU} and is equipped with an edge server e_j through a wired connection. The edge servers are heterogeneous in terms of processing and storage capabilities. A vehicle v_h can communicate with an edge server e_j only if it is under the communication range of RSU_j. We define a binary variable $\alpha_{v_h}^{e_j}(t) \in \{0, 1\}$; such that $\alpha_{v_h}^{e_j}(t) = 1$ means that v_h is in the range of RSU_j and can communicate with e_j and $\alpha_{v_h}^{e_j}(t) = 0$ otherwise. The third layer (i.e., cloud computing) consists of K heterogeneous cloud servers. The processing and storage capabilities of a cloud server c_k $(k \in \mathcal{K})$ is higher compared to that of an edge server e_j, $\forall j \in \mathcal{J}$, i.e., $\mu_{e_j} \ll \mu_{c_k}$ and $\theta_{e_j} \ll \theta_{c_k}$.

Each edge server in our model receives a set of requests from the communicating vehicles. The server makes the offloading decision for each request r_i, i.e., to execute the request locally on the edge e_j or to offload it to a cloud server c_k for execution such that the total execution time of all the requests is at the minimum while maintaining each request's latency, processing time, deadline, CPU, and memory SLA constraints. A binary variable $\beta_{r_i}^{s_z} \in \{0, 1\}$, $s_z \in \{e_j, c_k\}$ is defined such that, $\beta_{r_i}^{s_z} = 0$ if r_i is executed locally on the edge server e_j and $\beta_{r_i}^{s_z} = 1$ otherwise. For each offloaded request, the edge server sends the request and information about the cloud server c_k to the cloud manager. The cloud manager then schedules the request to c_k. Since both communication and computation are critical for making the offloading decision, we next introduce the communication and computation models in detail. Table 2 lists the notations used in this paper and their definitions.

Table 2
Notations and definitions.

Notation	Definition
$h,\ H,\ \mathcal{H}, v_h$	vehicle index, number of vehicles, set of vehicles, h^{th} vehicle
$i,\ I,\ \mathcal{I}, r_i,\ rep_i$	request index, number of requests, set of requests, i^{th} request, the reply of r_i
$j, J, \mathcal{J}, e_j,\ RSU_j$	edge server/RSU index, number of edge servers/RSUs, set of edge servers/RSUs, j^{th} edge server, j^{th} RSU
$k,\ K, \mathcal{K}, c_k$	cloud server index, number of cloud servers, set of cloud servers, k^{th} cloud server
$z, \mathcal{Z}, s_z$	server (edge/cloud) index ($z = \{j, k\}$), set of edge and cloud servers ($\mathcal{Z} = \{\mathcal{J}\} \cup \{\mathcal{K}\}$), z^{th} server $\left(s_z = \left\{e_j, c_k\right\}\right)$
ψ_{r_i}	length of r_i in Million Instructions (MI)
σ_{r_i}	size of r_i in kilobytes (KB)
φ_{r_i}	CPU utilization of r_i
$L_{r_i}^{max}$	maximum tolerable latency for r_i
$P_{r_i}^{max}$	maximum tolerable processing time for r_i
$D_{r_i}^{max}$	maximum tolerable deadline for r_i
$S_{v_h}(t)$	speed of v_h at time t
$\left(x_{v_h,r_i}^{src}, y_{v_h,r_i}^{src}\right)$	source location (longitude, latitude) of v_h while submitting r_i
$\left(x_{v_h,r_i}^{des}, y_{v_h,r_i}^{des}\right)$	destination location (longitude, latitude) of v_h that submitted r_i
D_{RSU}	the coverage area of each RSU
μ_{s_z}	processing speed of s_z in Million Instructions per Second (MIPS)
θ_{s_z}	available memory of s_z in KB
$\alpha_{v_h}^{e_j}(t)$	Whether or not v_h is in the communication range of RSU_j that is equipped with e_j
$\beta_{r_i}^{s_z}$	whether r_i is executed locally on e_j or offloaded to c_k
$T_{r_i(s_z)}^{com}$	total communication time of r_i when executed on s_z
$T_{r_i(x,y)}^{com}$	communication time to transfer r_i from x to y, where $x \in \left\{v_h,\ e_j\right\}$ and $y \in \left\{e_j, c_k\right\}$
$T_{rep_i(x,y)}^{com}$	communication time to transfer rep_i from x to y, where $x \in \left\{e_j, c_k, e_{j+y}\right\}$ and $y \in \left\{v_h,\ c_k,\ e_{j+y}\right\}$
$T_{r_i(s_z)}^{proc}$	processing time of r_i when executed on s_z
$T_{r_i(s_z)}^{I/O}$	I/O time of r_i when executed on s_z
$T_{r_i(s_z)}$	total execution time of r_i when executed on s_z
$\omega_{x,y}$	bandwidth between x and y in gigabits per second (Gbps), where $x, y \in \left\{v_h, e_j, c_k, e_{j+y}\right\}$
d_{v_h,e_j}	distance traveled by v_h in the communication range of e_j before submitting r_i
$d_{v_h}^{r_i-rep_i}(t)$	total distance traveled by v_h after submitting r_i and before receiving rep_i
$\tilde{d}_{v_h}^{r_i-rep_i}(t)$	distance traveled by v_h outside the range of e_j, after submitting r_i and before receiving rep_i
$\left(x_{e_j}^{left}, y_{e_j}^{left}\right)$	a point on the coverage boundary of RSU_j such that $x_{e_j}^{left} > x_{v_h,r_i}^{src}$ and $y_{e_j}^{left} = y_{v_h,r_i}^{src}$
$\left(x_{e_j}^{right}, y_{e_j}^{right}\right)$	a point on the coverage boundary of RSU_j such that $x_{e_j}^{right} < x_{v_h,r_i}^{src}$ and $y_{e_j}^{right} = y_{v_h,r_i}^{src}$
$\tau_{r_i(s_z)}^{m}$	processing time of r_i when overlapped with other requests on s_z
$\tau_{r_i(s_z)}^{a}$	processing time of r_i after other overlapping requests on s_z has finished execution
$n_{r_i(s_z)}$	number of requests being executed along with r_i on s_z
$\overline{n}_{r_i(s_z)}$	number of requests that were overlapping r_i on s_z and has completed execution before r_i
$\chi_{r_i(s_z)}$	number of times request data for r_i is swapped between disk and memory on s_z
ξ_{s_z}	time required to transfer data between disk and memory in s_z
$\rho_{r_i(s_z)}$	the ratio of memory required by r_i and the available memory on s_z
$q,\ \mathcal{Q}$	offloading solution index, a set of offloading solutions in a generation of genetic algorithm
$\ddot{F}_q, \ddot{\tilde{F}}_q$	non-penalized fitness score of q, normalized non-penalized fitness score of q
P_q^{lat}	latency violation of q
P_q^{proc}	processing time violation of q
$P_q^{deadline}$	deadline violation of q
P_q^{cpu}	CPU violation of q
P_q^{mem}	memory violation of q
$\tilde{P}_q$	normalized SLA violations of q
$\overline{F}_q,\ F_q$	adaptive penalized fitness score of q, fitness score of q
n_f	number of feasible offloading solutions in a generation
γ	the ratio of feasible to the total offloading solutions in a generation
P_{size}	the population size, i.e., the number of offloading solutions
$Cumm(q)$	cumulative fitness probability of q
$Prob(q)$	fitness probability of q
$\lambda_c,\ \lambda_m$	crossover rate, mutation rate
n_{mut}	number of requests for which the server allocation is mutated
G	number of generations

3.1. Communication model

The communication time in our model consists of the time required to transmit a request data from v_h to a scheduled server (based on an offloading decision) and the time required to transmit the reply back to v_h. Depending on the offloading decision (i.e., whether the request is executed on an edge or cloud server) and speed of v_h, there exist three scenarios as shown in Fig. 2:

- ***Scenario (a):*** v_h is communicating with RSU_j and r_i is submitted to e_j for execution. After r_i is executed, v_h is still in the communication range of RSU_j (Fig. 2 (a)). Consequently, in this scenario, the communication time involves: (1) the time to transfer the request from v_h to e_j and (2) the time to transfer the reply from e_j to v_h.
- ***Scenario (b):*** v_h is communicating with RSU_j and r_i is submitted to e_j for execution. After r_i is executed, v_h is in the range of RSU_{j+y} (Fig. 2 (b)). Consequently, in this scenario, the communication time includes: (1) the time to transfer the request from v_h to e_j, (2) the time to transfer the reply from e_j to cloud server c_k, (3) the time to transfer the reply from c_k to e_{j+y}, and (4) the time to

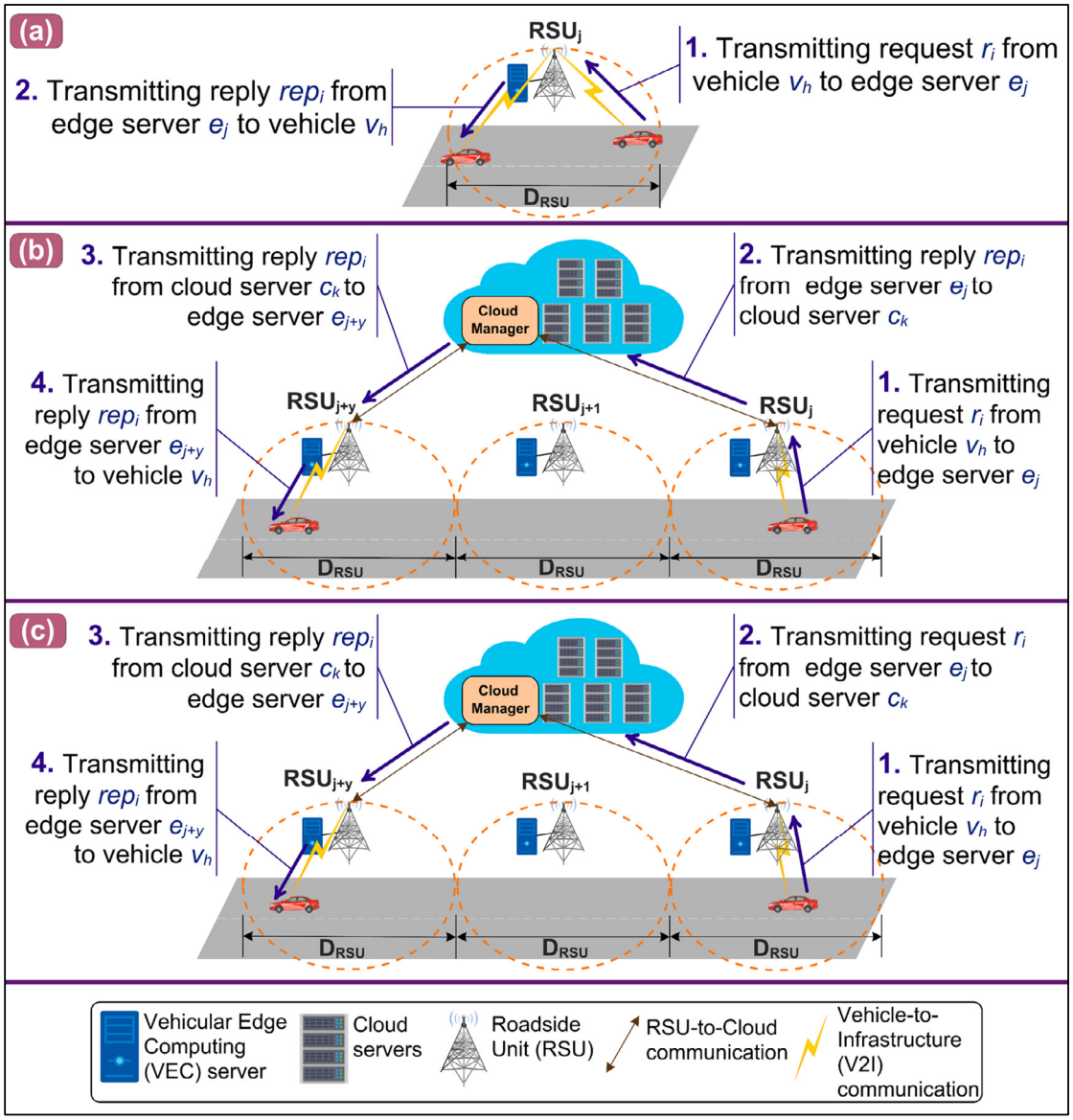


Fig. 2. Calculation of communication time based on offloading decision and vehicle speed. (a) Vehicle request is executed on an edge server and the vehicle is in the edge server's range when a reply on the request is received, (b) vehicle request is executed on an edge server and the vehicle moves out of the edge server's range before receiving a reply on the request, and (c) vehicle request is executed on a cloud server.

transfer the reply from e_{j+y} to v_h [30]. The data transmission between e_j and e_{j+y} is achieved via the cloud instead of multi-hop RSU transmission in our system model. This is because multi-hop RSU transmission is performed at a low rate which increases the response time of the request [31].

- ***Scenario (c):*** v_h is communicating with RSU_j and r_i is offloaded to the cloud and executed on a cloud server c_k (Fig. 2). After r_i is executed, v_h is in the range of RSU_{j+y}. Consequently, in this scenario, the communication time includes: (1) the time to transfer the request from v_h to e_j, (2) the time to transfer the requests from e_j to c_k, (3) the time to transfer the reply from c_k to e_{j+y}, and (4) the time to transfer the reply from e_{j+y} to v_h. In (Fig. 2 (c)), v_h moves out of RSU_j's communication range before receiving a reply. In case the vehicle is in the range of RSU_j after r_i is executed, the reply will be transmitted from c_k to e_j.

Based on these scenarios, the total communication time for a request r_i when executed on a server $s_z \mid s_z \in \{e_j, c_k\}$ can be computed as stated in (1).

$$T^{com}_{r_i(s_z)} = \begin{cases} T^{com}_{r_i(v_h,e_j)} + T^{com}_{rep_i(e_j,v_h)}; & (s_z = e_j) \text{ and } \left(\alpha^{e_j}_{v_h} = 1\right) \text{ while receiving the reply} \\ T^{com}_{r_i(v_h,e_j)} + T^{com}_{rep_i(,e_j,c_k)} + T^{com}_{rep_i(c_k,e_{j+y})} + T^{com}_{rep_i(e_{j+y},v_h)}; & (s_z = e_j) \text{ and } \left(\alpha^{e_j}_{v_h} = 0\right) \text{ while receiving the reply} \\ T^{com}_{r_i(v_h,e_j)} + T^{com}_{r_i(,e_j,c_k)} + T^{com}_{rep_i(c_k,e_{j+y})} + T^{com}_{rep_i(e_{j+y},v_h)}; & s_z = c_k \end{cases} \quad (1)$$

The communication times represented in (1) can be computed using (2)–(7).

$$T^{com}_{r_i(v_h,e_j)} = \frac{\sigma_{r_i}}{\omega_{v_h,e_j}} \tag{2}$$

$$T^{com}_{rep_i(e_j,v_h)} = \frac{\sigma_{rep_i}}{\omega_{e_j,v_h}} \tag{3}$$

$$T^{com}_{rep_i(e_j,c_k)} = \frac{\sigma_{rep_i}}{\omega_{e_j,c_k}} \tag{4}$$

$$T^{com}_{rep_i(c_k,e_{j+y})} = \frac{\sigma_{rep_i}}{\omega_{c_k,e_{j+y}}} \tag{5}$$

$$T^{com}_{rep_i(e_{j+y},v_h)} = \frac{\sigma_{rep_i}}{\omega_{e_{j+y},v_h}} \tag{6}$$

$$T^{com}_{r_i(e_j,c_k)} = \frac{\sigma_{r_i}}{\omega_{e_j,c_k}} \tag{7}$$

In scenarios (b) and (c), the mobile vehicle will move out of RSU_j's communication range and will be in the communication range of RSU_{j+y} while receiving the request's reply. Consequently, to deliver the reply to the vehicle and to compute the communication time in scenarios (b) and (c), the cloud manager should determine RSU_{j+y}, i.e., y^{th} RSU after RSU_j, which is located on the path between v_h's source and destination. The path between the source and destination can be computed by the cloud offline using an extended A* algorithm [32]. The selection of the extended A* algorithm is based on its performance compared to the shortest path algorithm. The algorithm determines an optimal path between the source and destination in a way that reduces the vehicle's fuel consumption. The algorithm begins by dividing driving mode, through a signalized intersection, into four driving conditions. This is based on the current state of traffic lights. Then, it calculates the probability for each driving condition. These conditions are described below.

1. Driving through the intersection at a constant speed: The state of traffic light approached by the vehicle is green and will remain the same while the vehicle is passing through the intersection while maintaining its current speed.
2. Driving through the intersection by accelerating: The state of traffic light approached by the vehicle is green and will change to red in a short time. Consequently, the vehicle should accelerate to the maximum speed limit to pass through the intersection before the end of the green phase. After crossing the intersection, the vehicle decelerates from the maximum speed to the average road speed.
3. Driving through the intersection by decelerating: The state of traffic light approached by the vehicle is red and will change to green in a short time. Consequently, the vehicle should decelerate to the minimum speed limit to pass through the intersection after the end of the red phase. The vehicle accelerates to the average road speed after passing the intersection.
4. Waiting at the intersection: The state of traffic light approached by the vehicle is red and will remain the same for a long time. Consequently, the vehicle should decelerate and stop at the intersection till the traffic light becomes green. The vehicle accelerates to the average road speed after passing the intersection once the phase of the signal is green.

The path between the source and destination is determined in a way that the fuel consumption of a vehicle is the minimum. The fuel consumption rate of the vehicle for each driving condition is calculated based on acceleration and velocity data.

As shown in Fig. 3, the cloud manager will determine the y^{th} RSU after RSU_j $\left(i.e.,\ RSU_{j+y}\right)$ on the path between the vehicle's source and destination based on the vehicle's speed, the vehicle's position, and the request's execution time. The value of y^{th} RSU is determined as described below:

- Each vehicle transmits its speed to the communication edge server. The server further sends this information to the cloud manager.
- The cloud manager computes the distance v_h had traveled in the communication range of RSU_j before submitting request r_i, i.e., d_{v_h,e_j}, as stated in Equation (8).

$$d_{v_h,e_j} = \begin{cases} x^{left}_{e_j} - x^{src}_{v_h,r_i}; & x^{src}_{v_h,r_i} > x^{des}_{v_h,r_i} \\ x^{src}_{v_h,r_i} - x^{right}_{e_j}; & x^{src}_{v_h,r_i} < x^{des}_{v_h,r_i} \end{cases} \tag{8}$$

- Depending on the vehicle's speed and the request's total execution time, the manager will dynamically compute the total distance v_h will travel after submitting r_i to RSU_j and before receiving rep_i, i.e., $d^{r_i-rep_i}_{v_h}(t)$, as stated in (9).

$$d^{r_i-rep_i}_{v_h}(t) = s_{v_h}(t) \times T_{r_i(s_z)} \tag{9}$$

- The cloud manager will now calculate $\tilde{d}^{r_i-rep_i}_{v_h}(t)$ which is the distance v_h will travel outside the range of RSU_j, after submitting r_i and before receiving rep_i, as stated in Equation (10).

$$\tilde{d}^{r_i-rep_i}_{v_h}(t) = \begin{cases} d^{r_i-rep_i}_{v_h}(t) - \left(D_{RSU} - d_{v_h,e_j}\right); & d_{v_h,e_j} + d^{r_i-rep_i}_{v_h}(t) > D_{RSU} \\ 0; & otherwise \end{cases} \tag{10}$$

- The value of y^{th} RSU is then computed using Equation (11). The value of y will be updated in real-time based on the speed of v_h.

$$y(t) = \left\lceil \frac{\tilde{d}^{r_i-rep_i}_{v_h}(t)}{D_{RSU}} \right\rceil \tag{11}$$

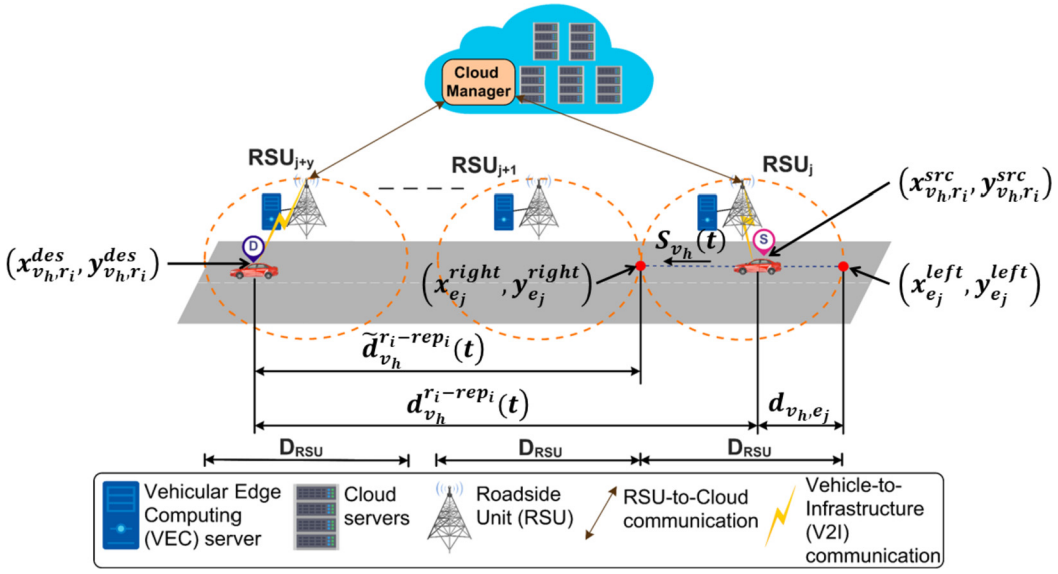


Fig. 3. Calculation of y^{th} RSU, on the path between the vehicle's source and destination, to send a reply to the request.

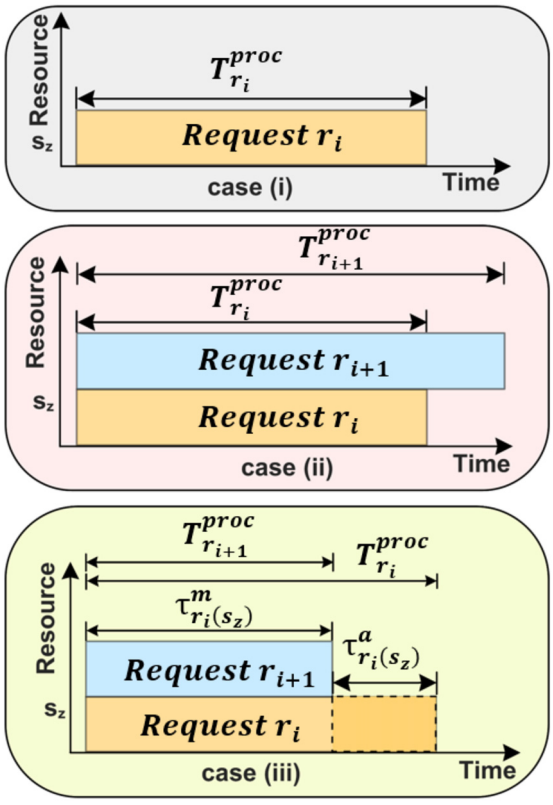


Fig. 4. Calculation of processing time in a multi-request scenario.

3.2. Computation model

The computation time in our system model includes the processing time to execute the request on an edge/cloud server and the I/O time required for the request data transfer between the memory and disk of an edge/cloud server. The processing time and I/O time are explained below in detail.

3.2.1. Processing model

The processing time of a request in our system model depends on whether the request is executed alone or with other requests on an edge/cloud server. Consequently, there exist three cases as shown in Fig. 4.

- ***Case (i):*** request r_i is executed alone on an edge/cloud server s_z.

- ***Case (ii):*** execution of requests r_i and r_{i+1} overlap on server s_z and r_i completes execution before r_{i+1}. In this case, the processing speed of s_z is divided among requests r_i and r_{i+1} [33].
- ***Case (iii):*** execution of requests r_i and r_{i+1} overlap on server s_z and r_i completes execution after r_{i+1}. In this case, the processing time of r_i includes the time when the execution of r_i and r_{i+1} overlaps (i.e., $\tau^m_{r_i(s_z)}$) and the time when r_i executes alone after r_{i+1} has finished execution (i.e., $\tau^a_{r_i(s_z)}$).

Depending on which server is selected for execution based on the offloading decision, the processing time of the request can be computed as stated in (12).

$$T^{proc}_{r_i(s_z)} = \begin{cases} \left(\frac{\psi_{r_i}}{\mu_{s_z}}\right); & case\ (i) \\ \left(\frac{\psi_{r_i} \times n_{r_i(s_z)}}{\mu_{s_z}}\right); & case\ (ii) \\ \tau^m_{r_i(s_z)} + \tau^a_{r_i(s_z)}; & case\ (iii) \end{cases} \tag{12}$$

where $\tau^m_{r_i(s_z)}$, $\tau^a_{r_i(s_z)}$, and $\overline{n}_{s_z}$ can be calculated using (13)–(15).

$$\tau^m_{r_i(s_z)} = \min_{\forall r_p \in s_z} \left(T^{proc}_{r_p(s_z)}\right), \quad p \neq i \tag{13}$$

$$\tau^a_{r_i(s_z)} = \left(\frac{\psi_{r_i} - \left(\frac{\tau^m_{r_i(s_z)} \times \mu_{s_z}}{n_{r_i(s_z)}}\right)}{\mu_{s_z}}\right) \times \left(n_{r_i(s_z)} - \overline{n}_{r_i(s_z)}\right) \tag{14}$$

$$\overline{n}_{r_i(s_z)} = \underset{r_i, r_p \in s_z}{\#} \left(T_{r_p(s_z)} < T_{r_i(s_z)}\right), \quad i \neq p \tag{15}$$

3.2.2. I/O model

The I/O time for a request r_i on server s_z in our system model refers to the time it takes to transfer data between disk and memory in case the memory requirement of r_i is more than the available memory of s_z. The I/O time of r_i when executed on s_z for the three cases discussed before can be computed using (16).

$$T^{I/O}_{r_i(s_z)} = \begin{cases} \left(\chi_{r_i(s_z)} \times \xi_{s_z}\right); & cases\ (i)\ and\ (ii) \\ \xi_{s_z} + \frac{\sigma_{r_i} \times \xi_{s_z} \times \overline{n}_{r_i(s_z)}}{\theta_{s_z}}; & case\ (iii) \end{cases} \tag{16}$$

where $\chi_{r_i(s_z)}$ is calculated using (17) and (18) as follows.

$$\chi_{r_i(s_z)} = \begin{cases} \rho_{r_i(s_z)} - 1; & \rho_{r_i(s_z)} > 1 \\ 0; & \rho_{r_i(s_z)} = 1 \end{cases} \tag{17}$$

$$\rho_{r_i(s_z)} = \begin{cases} \left\lceil \frac{\sigma_{r_i}}{\theta_{s_z}} \right\rceil; & case\ (i) \\ \left\lceil \frac{\sigma_{r_i} \times n_{r_i(s_z)}}{\theta_{s_z}} \right\rceil; & cases\ (ii)\ and\ (iii) \end{cases} \tag{18}$$

4. Problem formulation

We formulate a computation offloading optimization problem in an integrated edge-cloud computing system for IoV. The objective of the optimization problem is to minimize the total execution time of all the requests in the system under specified latency, processing time, deadline, CPU, and memory requirements constraints for each request. The total execution time of a request r_i is computed as the summation of the request's communication, processing, and I/O times as stated in (19). To this end, the corresponding optimization problem can be formulated as stated in (20).

$$T_{r_i(s_z)} = T^{com}_{r_i(s_z)} + T^{proc}_{r_i(s_z)} + T^{I/O}_{r_i(s_z)}, \quad \forall i \in \mathcal{I}, s_z \in \{e_j, c_k\} \tag{19}$$

Problem:

$$minimize \sum_{\forall i \in \mathcal{I}} T_{r_i(s_z)}, \quad s_z \in \{e_j, c_k\} \tag{20}$$

$$\textbf{s.t.}\quad \text{C1:}\quad \forall i \in \mathcal{I} T^{com}_{r_i(s_z)} \leq L^{max}_{r_i},\ s_z \in \left\{e_j, c_k\right\}$$

$$\text{C2:}\quad \forall i \in \mathcal{I} T^{proc}_{r_i(s_z)} \leq P^{max}_{r_i},\ s_z \in \left\{e_j, c_k\right\}$$

$$\text{C3:}\quad \forall i \in \mathcal{I} T_{r_i(s_z)} \leq D^{max}_{r_i},\ s_z \in \left\{e_j, c_k\right\}$$

$$\text{C4:}\quad \sum_{\forall r_i \in s_z} \varphi_{r_i} \leq \varphi^{max}_{s_z},\ s_z \in \left\{e_j, c_k\right\}\ and\ \varphi_{r_i} \geq 1$$

$$\text{C5:}\quad \sum_{\forall r_i \in s_z} \sigma_{r_i} \leq \theta_{s_z},\ z \in \mathcal{Z}$$

$$\text{C6:}\quad \alpha^{e_j}_{v_h}(t) \in \{0, 1\}, \forall j \in \mathcal{J},\ \forall h \in \mathcal{H}$$

$$\text{C7:}\quad \sum_{j \in \mathcal{J}} \alpha^{e_j}_{v_h}(t) = 1, \forall h \in \mathcal{H}$$

$$\text{C8:}\quad \sum_{\forall z \in \mathcal{Z}} \beta^{s_z}_{r_i} = 1, \forall i \in \mathcal{I}$$

where $\sum_{\forall i \in \mathcal{I}} T_{r_i(s_z)}$ is the sum of the total execution time of all the requests in the system. The constraints in the above optimization problem are as follows:

- C1 ensures that the communication time of each request does not exceed the maximum tolerable latency requirement of that request.
- C2 guarantees that the execution time of each request is less than the request's permissible maximum processing time requirement.
- C3 ensures that the total execution time of each request is below the maximum tolerable deadline for that request.
- C4 ensures that the total CPU utilization of all the requests that are being executed simultaneously on a server should not exceed the server's CPU utilization threshold. This is to ensure that the server is not overloaded as overloading may degrade the requests' performances.
- C5 ensures that the memory requirements of a request should be less than the server's available memory. This is to reduce the amount of data transfer between disk and memory.
- C6 and C7 denote that each vehicle can only communicate with one edge server at a given time.
- C8 ensures that each request is executed at most by only one edge/cloud server.

Theorem 1. *The optimization problem in (20) is NP-hard.*

Proof. To prove the NP-hardness of the offloading problem, we first consider a special case of the problem where the SLA requirements of the requests in terms of latency, processing time, and deadline, i.e., C1 – C3 in (20), are not violated. Consequently, the offloading problem is rendered to minimization of the sum of the total execution times for all requests. We prove that the special case of our offloading problem is NP-hard by reducing a known NP-hard problem, i.e., multiple knapsack problem [34], to the special case of our offloading problem in a polynomial time. We denote this by (21).

$$(Multiple\ knapsack) \leq p\left(Offloading_{special\ case}\right) \tag{21}$$

Multiple knapsack is a well-known NP-hard problem [35], where given a set of knapsacks, each having a certain capacity, the objective is to assign disjoint subsets of items (each item having a weight and a value) to a unique knapsack such that the total value of items is maximized while the total weight of items in each knapsack should not exceed the knapsack's capacity. For any instance of the multiple knapsack problem, an instance of the offloading problem's special case can be constructed where the vehicles' requests represent the items, and the edge and cloud servers represent the knapsacks. The request's resource requirements $\left(\psi_{r_i}, \sigma_{r_i}, \varphi_{r_i}\right)$ represents the weight and the total execution time represents a value. For an edge or cloud server, its threshold CPU utilization $\varphi^{max}_{s_z}$ and available memory θ_{s_z} represent the capacity. Then, filling the items to knapsacks is equivalent to assigning the requests to the edge/cloud servers such that the sum of the total execution time of all the requests is minimized, and the $\varphi^{max}_{s_z}$ and θ_{s_z} requirements of the servers are satisfied.

As the NP-hard multiple knapsack problem is reduced to the special case of the offloading problem, the special case is an NP-hard problem as well. Consequently, it can be inferred that the offloading problem is also NP-hard. Therefore, we propose an adaptive genetic algorithm to obtain the optimal solution in polynomial time.

5. Proposed QoS-SLA-aware adaptive genetic offloading algorithm

The proposed offloading aims to minimize the total execution time of the vehicles' requests while respecting each request's SLA requirements. In this paper, we propose a QoS-SLA-AGA to obtain the solution of the NP-hard offloading algorithm. GA [36], [37] is based on the theory of natural evolution where a subset of near-optimal offloading solutions from one generation is used to obtain the offspring solution for the next generation. At each generation, the algorithm converges towards the global optima. In the context of our optimization problem, global optima can be defined as an offloading solution that results in the minimum requests execution time while respecting each request's SLA requirements. An offloading solution, referred to as chromosome in genetic algorithm terminology, consists of server allocation for each request. The length of a chromosome is the same as the number of requests to be allocated. Each request-server

Algorithm 1: QoS-SLA-Aware Adaptive Genetic Offloading Algorithm.

Input: $H, I, J, K, r_i, e_j, c_k, P_{size}, \lambda_c, \lambda_m, q, n_{mut}$
Output: Scheduled requests on edge and cloud servers
/* **A. Initialization of offloading solutions** */
for $q = 1$ *to* P_{size} **do** /* for each offloading solution in the population */
  for $i = 1$ *to* I **do** /* for each request */
    $request_i \leftarrow r_i$
    $server_i \leftarrow selectRandom\left(\left\{e_j \mid \alpha_{v_h}^{e_j}(t) = 1\right\} \cup \{c_k \mid k \in \mathcal{K}\}\right)$ /* allocating a server randomly from a set of servers that consists of the edge server to which the request is submitted and the cloud servers */
  end for
  $offloading\_solution(q) \leftarrow tuple(requests, servers)$ /* assigning the requests to servers is an offloading solution in the population */
end for
$population \leftarrow tuple(offloading\_solutions)$ */ population consists of initialized offloading solutions */
repeat
/* **B. Evaluation of offloading solutions** */
  for $q = 1$ *to* P_{size} **do** /* for each offloading solution in the population */
    $\ddot{F}_q \leftarrow Equation\ (22)$ /* compute non-penalized fitness value */
    $P_q^{lat}, P_q^{proc}, P_q^{deadline}, P_q^{cpu}, P_q^{mem} \leftarrow Equations\ (23), (24), (25), (26), (27)$ /* compute latency, processing time, deadline, CPU, and memory requirements violations */
    $\tilde{\ddot{F}}_q \leftarrow Equation\ (28)$ /* compute normalized non-penalized fitness value */
    $\tilde{P}_q \leftarrow Equation\ (29)$ /* compute normalized penalty for constraints violations */
    $\bar{F}_q \leftarrow Equation\ (30)$ /* compute adaptive penalized fitness value */
    $F_q \leftarrow Equation\ (31)$ /* compute fitness value */
  end for
/* **C. Selection of offloading solutions to reproduce solutions for the next generation** */
  for $q = 1$ *to* P_{size} **do** /* for each offloading solution in the population */
    $Prob(q) \leftarrow Equation\ (33)$ /* computing fitness probability for each solution */
    $Cumu(q) \leftarrow Equation\ (32)$ /* computing cumulative fitness probability for each solution */
    $rnd(q) \leftarrow GenRandomNum(0, 1)$ /* generating a random number for each solution */
  end for
  for $k = 1$ *to* P_{size} **do** /* for each solution in the population */
    $selected\_solution(k) \leftarrow find(Cumu > rnd(k))$ /* select a solution such that the random number lies between cumulative probability bounds for that solution */
  end for
/* **D. Crossover operation to develop solutions for the next generation** */
  $pairs\_list \leftarrow GeneratePairs(selection\_solution)$ /* generate pairs of offloading solutions from the selected solutions */
  for each $pair \in pairs\_list$ **do**
    select $parent\_solution_1$ *and* $parent\_solution_2$ *from each pair* /*define parent solutions for each pair */
    $cutoff \leftarrow GenRandomNum(1, I)$ /*generate a random cutoff value between 1 and the length of each solution. The length of a solution is the number of requests */
    $t \leftarrow 1$ /* initialize a variable */
    for $1 \leq t < cutoff$ **do** /* for tasks before the cutoff value */
      $offspring\_solution_1(t) \leftarrow parent\_solution_1(t)$ /* offspring 1 will have request-server mapping similar to parent 1 */
      $offspring\_solution_2(t) \leftarrow parent\_solution_2(t)$ /* offspring 2 will have request-server mapping similar to parent 2 */
    end for
    for $cutoff < t \leq I$ **do** /* for tasks after the cutoff value */
      $offspring\_solution_1(t) \leftarrow parent\_solution_2(t)$ /* offspring 1 will have request-server mapping similar to parent 2 */
      $offspring\_solution_2(t) \leftarrow parent\_solution_1(t)$ /* offspring 2 will have request-server mapping similar to parent 1 */
    end for
    $solution(pair) \leftarrow Best(parent\_solution_1, parent\_solution_2, offspring\_solution_1, offspring\_solution_2)$ /* selecting the best two solutions among parents and offspring for next-generation */
  end for
/* **E. Mutation operation to diversify offloading solutions in a generation** */
  for $i = 1$ *to* n_{mut} **do** /* for the total number of mutation operations */
    $rnd(i) \leftarrow GenRandomNum(1, (I \times P_{size}))$ /* generate a random number between 1 and total requests in the entire population */
    if $rnd(i) \mod I = 0$ **then** /* if the remainder after dividing the random number with the total number of requests is zero */
      $solution_i \leftarrow \left(\frac{rnd(i) - (rnd(i) \mod I)}{I}\right)$ /* finding the solution where the random number lies */
      $request\_to\_reallocate_i \leftarrow I$ /* finding the request for reallocation for that solution */
    else /* if the remainder after dividing the random number with the total number of requests is not zero */
      $solution_i \leftarrow \left(\frac{rnd(i) - (rnd(i) \mod I)}{I}\right) + 1$
      $request\_to\_reallocate_i \leftarrow rem_i$
    end if
    $Solution_i(request\_to\_reallocate_i) \leftarrow selectRandom\left\{e_j, c_k \mid \alpha_{v_h}^{e_j}(t) = 1, k \in \mathcal{K}\right\}$ /* randomly reallocating the server for the selection request */
  end for
until the termination condition is satisfied

allocation is known as a gene. The number of offloading solutions in each generation represents the population size (P_{size}) and remains constant throughout the generations. In the following, we explain the steps involved in our proposed algorithm. Algorithm 1 shows its pseudocode.

5.1. Initialization of offloading solutions

QoS-SLA-AGA begins with a set random set of offloading solutions. In this step, a set, $\mathcal{Q}$, of initial offloading solutions, for the first generation, is randomly developed to begin the exploration process in the search space. The number of solutions in the set is equal to P_{size}. The value of P_{size} should be carefully selected as it impacts the convergence of the algorithm. A small value improves the computational performance of the algorithm, however, may restrict the search space leading to local optima instead of global. On the other hand, a large value allows the algorithm to explore a larger search space that might lead to global optima. However, this increases the computational time.

5.2. Evaluation of the offloading solutions

In this step, each offloading solution generated in the previous step is evaluated, in terms of fitness, to determine how close it is to the optimal offloading solution. The closer a solution is to the optimal solution, the higher its fitness. Consequently, based on our optimization objective function stated in (20), an offloading solution having the least total execution time for all requests with no SLA violations for each request will have the highest fitness value. The offloading solutions that violate SLA requirements are referred to as infeasible solutions. To incorporate the SLA constraints in fitness computation, we implement an adaptive penalty function [38] that reduces the fitness value of an infeasible solution. To evaluate the fitness with an adaptive penalty, we first compute the non-penalized fitness value as stated in (22) and SLA violations, in terms of latency, processing time, deadline, CPU utilization, and memory resource, for each solution as stated in (23)–(27).

$$\ddot{F}_q = \sum_{\forall i \in \mathcal{I}} T_{r_i(s_z)}, \quad s_z \in \{e_j, c_k\}, \quad \forall q \in \mathcal{Q} \tag{22}$$

$$P_q^{lat} = \sum_{\forall T^{com}_{r_i(s_z)} > L^{max}_{r_i}} T^{com}_{r_i(s_z)} - L^{max}_{r_i} \tag{23}$$

$$P_q^{proc} = \sum_{\forall T^{proc}_{r_i(s_z)} > P^{max}_{r_i}} T^{proc}_{r_i(s_z)} - P^{max}_{r_i} \tag{24}$$

$$P_q^{deadline} = \sum_{\forall T_{r_i(s_z)} > D^{max}_{r_i}} T_{r_i(s_z)} - D^{max}_{r_i} \tag{25}$$

$$P_q^{cpu} = \sum_{\forall r_i \in s_z} I_{(\varphi_{r_i} > \varphi^{max}_{s_z})}, \quad \forall z \in \mathcal{Z} \tag{26}$$

$$P_q^{mem} = \sum_{\forall r_i \in s_z} I_{(\sigma_{r_i} > \theta_{s_z})}, \quad \forall z \in \mathcal{Z} \tag{27}$$

The non-penalized fitness and the constraints violations are normalized using (28) and (29) respectively. The adaptive penalized fitness is then calculated as stated in (30).

$$\tilde{\ddot{F}}_q = \frac{\ddot{F}_q - \min_{\forall q \in \mathcal{Q}} \left(\ddot{F}_q\right)}{\max_{\forall q \in \mathcal{Q}} \left(\ddot{F}_q\right) - \min_{\forall q \in \mathcal{Q}} \left(\ddot{F}_q\right)} \tag{28}$$

$$\tilde{P}_q = \frac{1}{5}\left(\frac{P_q^{lat}}{\max_{\forall q \in \mathcal{Q}} \left(P_q^{lat}\right)} + \frac{P_q^{proc}}{\max_{\forall q \in \mathcal{Q}} \left(P_q^{proc}\right)} + \frac{P_q^{deadline}}{\max_{\forall q \in \mathcal{Q}} \left(P_q^{deadline}\right)} + \frac{P_q^{cpu}}{\max_{\forall q \in \mathcal{Q}} \left(P_q^{cpu}\right)} + \frac{P_q^{mem}}{\max_{\forall q \in \mathcal{Q}} \left(P_q^{mem}\right)}\right) \tag{29}$$

$$\overline{F}_q = \begin{cases} \tilde{P}_q; & n_f = 0 \\ \tilde{\ddot{F}}_q; & \tilde{P}_q = 0 \\ \sqrt{\left(\tilde{\ddot{F}}_q\right)^2 + \left(\tilde{P}_q\right)^2} + \left[(1-\gamma)\,\tilde{P}_q + (\gamma)\,\tilde{\ddot{F}}_q\right]; & otherwise \end{cases} \tag{30}$$

The fitness score for each solution is then computed by taking the reciprocal of adaptive penalized fitness as stated in (31). This is to assign the highest fitness value to the offloading solution having the least execution time and QoS violations.

$$F_q = \frac{1}{\overline{F}_q + 1} \tag{31}$$

5.3. Selection of offloading solutions to reproduce solutions for next generation

In this step, offloading solutions from the population are selected based on their fitness value to reproduce offspring offloading solutions for the next generation. In this paper, we use the fitness proportionate Roulette Wheel Selection (RWS) [39] method that constructs a roulette wheel based on the cumulative fitness probabilities of the offloading solutions. The fittest a solution is, the larger the area

Table 3
Computational complexity of the proposed algorithm.

Algorithm step	Computational complexity
Initialization of offloading solutions	$O\left(P_{size} \times I\right)$
Evaluation of the offloading solutions	$O\left(G \times P_{size} \times \left(I + Zn^2_{r_i(s_z)}\right)\right)$
Selection of offloading solutions to reproduce solutions for next generation	$O\left(G \times P_{size}\right)$
Crossover to develop solutions for next generation	$O\left(G \times \left(P_{size} + Zn_{r_i(s_z)} + (I \times \lambda_c)\right)\right)$
Mutation to diversify offloading solutions in a generation	$O\left(G \times P_{size} \times I \times \lambda_m\right)$
Total	$O\left(G \times \left(\left(P_{size} \times Zn_{r_i}\right) + (I \times \lambda_c) + (P_{size} \times I \times \lambda_m)\right)\right)$

Note: $Z = K + 1$.

occupied by that solution on the roulette wheel. The cumulative probability for each offloading solution can be computed using (32). Offloading solutions are then selected based on the position of randomly generated numbers on the roulette wheel.

$$Cumu\left(q\right) = \sum_{l=1}^{q} Prob(l) \tag{32}$$

Where $Prob\left(q\right)$ represents the fitness probability of offloading solution q $\left(q \in \mathcal{Q}\right)$ and can be computed using (33).

$$Prob(q) = \frac{F_q}{\sum_{\forall q \in \mathcal{Q}} F_q} \tag{33}$$

5.4. Crossover to develop offloading solutions for next generation

In this step, the selected fit offloading solutions are used to produce offspring solutions by swapping the request-server allocations for two offloading solutions, known as parent solutions. Crossover operation produces fitter offspring offloading solutions from fit parent solutions leading to convergence of the algorithm towards the optimal solution. The number of parent solutions selected for crossover depends on the crossover rate λ_c. We use a single-point crossover where a cutoff point for crossover is generated randomly and all the server allocations for the requests after the cutoff point from the parents are swapped resulting in two offspring solutions. Parent solutions in the generation are then replaced by the two fittest solutions among the parent and offspring solutions.

5.5. Mutation to diversify the offloading solutions in a generation

In this step, the offloading decisions for some requests in the population are changed to diversify the offloading solutions and larger the search space. Without mutation, the algorithm may converge prematurely, i.e., on the local optima, as the search space would be restricted around the non-optimal fit solutions in the population. The number of requests for which the offloading decisions are changed depends on the mutation rate parameter and can be calculated using (34).

$$n_{mut} = I \times P_{size} \times \lambda_m \tag{34}$$

5.6. Termination of the algorithm

In this step, the algorithm is terminated if the maximum number of user-defined generations is reached, or the optimal offloading solution is obtained. The evaluation, selection, crossover, and mutation steps are iterated until termination.

5.7. Complexity and convergence analysis

Table 3 presents the computational complexity of the proposed QoS-SLA-AGA. It shows that the complexity mainly depends on the population size, the number of generations till convergence, and the number of servers in edge and cloud data centers. The number of requests is an adding factor to the complexity, and it depends on the traffic flow.

In summary, the search space of offloading solutions increases with growing edge and cloud data centers sizes and traffic flow. However, the proposed algorithm converges to an optimal offloading solution as it satisfies the accessibility and absorption properties mentioned below.

Property 1. *For any generation g, if $Pop(g) \in S_{opt}^{P_{size}}$, then $Pop_s\left(g+1\right) \in S_{opt}^{P_{size}}$. This property is called the absorption strategy because the offloading solution with the highest fitness value is always kept during evolution. Where, $Pop(g)$ is the population state, i.e., set of offloading solutions, at the g^{th} generation $(g = 1,\ 2,\ \ldots,\ G)$, $S_{opt}^{P_{size}}$ is the global optimal set of offloading solutions as presented in Equation (35), and $Pop_s\left(g+1\right)$ is the population state after the selection of offloading solutions at $g + 1^{th}$ generation.*

$$\mathrm{S}_{\mathrm{opt}}^{\mathrm{P}_{\mathrm{size}}} = \left\{\mathcal{Q}; \exists q \in \mathcal{Q},\ q \in S_{opt}\right\} \tag{35}$$

where $\mathrm{S}_{\mathrm{opt}}$ is the optimal offloading solution as stated in Equation (36).

$$\mathrm{S}_{\mathrm{opt}} = \left\{q, q \in \mathcal{Q},\ F_q = F_{max}\right\} \tag{36}$$

where F_{max} is the theoretical optimal fitness value.

Property 2. *For any generation g, if* $Pop_s(g) \in S_{opt}^{P_{size}}$, *then* $Pop_c(g+1) \in S_{opt}^{P_{size}}$, *where* $Pop_c(g)$ *is the population state after performing crossover at* $g+1^{th}$ *generation.*

Property 3. *For any generation g, if* $Pop_c(g) \in S_{opt}^{P_{size}}$, *then* $Pop_m(g+1) \in S_{opt}^{P_{size}}$, *where* $Pop_m(g)$ *is the population state after performing mutation at* $g+1^{th}$ *generation.*

Property 4. *There exists some generation* g_0, *for any initial population* $Pop(1) = \mathcal{Q}$, *any measurable subset* $A \subseteq \mathcal{Q}^{P_{size}}$ *and some* $\delta(\mathcal{Q}) > 0$, *the transition probability function satisfies* $P(1, g_0; \mathcal{Q}, A) \geq \delta(\mathcal{Q})\phi(A)$. $\phi(.)$ *is a measure in a set of offloading solutions* $\mathcal{Q}$ *and* $\mathcal{Q}^{P_{size}} = \mathcal{Q} \times \mathcal{Q} \times \mathcal{Q} \times \cdots \times \mathcal{Q}$ *is the product space, i.e., population space, that contains all possible offloading solutions for a given optimization problem, i.e., the entire search space which constitutes all the generations. This property is called the accessibility strategy because starting from any initial set of offloading solution* $\mathcal{Q}$, *the global optimal set is accessible after finite probability transitions.* $\mathcal{Q} = \{1, 2, \ldots, q, \ldots, P_{size}\}$ *is a measurable space of offloading solutions in a generation, i.e., population. This space is assumed to be bounded and compact.*

In the following, we prove the convergence of QoS-SLA-AGA using the Markov chain theory and the above properties [40], [41]. Based on Equation (31), our objective is to maximize the fitness function $\left(F_q; q \in \mathcal{Q}\right)$ of an offloading solution q. The fitness function for a set of offloading solutions in a generation can be defined as stated in Equation (37).

$$\mathrm{F}(\mathcal{Q}) = \max\left\{F_q;\ q \in \mathcal{Q}\right\} \tag{37}$$

Let us consider that $Pop(g)$, $Pop_s(g)$, $Pop_c(g)$, $Pop_m(g)$, and $Pop(g+1)$ are all random variables in the population space $\mathcal{Q}^{P_{size}}$. The evolution of population states in QoS-SLA-AGA can thus be represented as stated in Equation (38).

$$\mathrm{Pop}(g) \rightarrow Pop_s(g) \rightarrow Pop_c(g) \rightarrow Pop_m(g) \rightarrow Pop(g+1) \tag{38}$$

In QoS-SLA-AGA, the selection, crossover, and mutation operators are time-invariant. Consequently, the transition probability functions for these operators are time-invariant. The transition probability functions for selection, crossover, and mutation are presented in Equations (39)–(41) respectively.

$$P_s(\mathcal{Q}, A) = P_s\left(Pop_s(g) \in A \mid Pop(g) = \mathcal{Q}\right) \tag{39}$$

$$P_c(\mathcal{Q}, A) = P_c\left(Pop_c(g) \in A \mid Pop_s(g) = \mathcal{Q}\right) \tag{40}$$

$$P_m(\mathcal{Q}, A) = P_m\left(Pop_m(g) \in A \mid Pop_c(g) = \mathcal{Q}\right) \tag{41}$$

where $A \subseteq \mathcal{Q}^{P_{size}}$

The proposed QoS-SLA-AGA can be modeled by a stationary Markov chain $\left\{Pop(g); g \in Z^+\right\}$ on the state space $\mathcal{Q}^{P_{size}}$, whose transition probability is given by the Kolmogorov-Chapman equation as stated in Equation (42).

$$\mathrm{P}(\mathcal{Q}, A) = \iint Pop_s(\mathcal{Q}, dy)\, Pop_c(y, dz)\, Pop_m(z, A) \tag{42}$$

Based on this modeling, our proposed algorithm converges to the global optimal as indicated by Theorem 2.

Theorem 2. *Assume that* $\mathcal{Q}^{P_{size}}$ *is a measurable space that is bounded and compact, and* $\left\{Pop(g); g \in Z^+\right\}$ *is the Markov chain given by Equation (42). If the Markov chain satisfies Properties 1, 2, 3, and 4, then it converges to the global optimal set.*

Proof. From Lemma 1 in [42], it is proved that the Markov chain satisfies Doeblin's condition which states that there is a probability measure $\partial(.)$ on the space $\mathcal{Q}^{P_{size}}$ and some generation $g_0 > 0$ and some positive $\delta > 0$, such that, for any set $A \subseteq \mathcal{Q}^{P_{size}}$, it holds $P(1, g_0; \mathcal{Q}, A) \geq \delta\partial(A)$. Consequently, the Markov chain $\left\{Pop(g); g \in Z^+\right\}$ converges in the rate $\left|\left|\mu_g - \pi\right|\right| \leq (1-\delta)^{\left[\frac{g}{g_0}\right]-1}$. Here, μ_g denotes the probability distribution of $Pop(g)$ and π is the probability distribution on $\mathcal{Q}^{P_{size}}$.

Furthermore, from Lemma 2 in [42], it is proved that for any invariant probability distribution π on $\mathcal{Q}^{P_{size}}$, $\pi\left(S_{opt}^{P_{size}}\right) = 1$. From Lemma 1 and Lemma 2, it is proved that $\left\{Pop(g); g \in Z^+\right\}$ converges to the global optimal set $S_{opt}^{P_{size}}$.

6. Performance evaluation

We analyze the impact of different λ_c, λ_m, and P_{size} on the convergence of our proposed algorithm and compare its performance with baseline approaches in terms of total execution time (seconds) and the number of requests violating SLA constraints.

6.1. Experimental environment

We created a heterogeneous integrated edge-cloud computing system for IoV. 10 edge servers and 20 cloud servers were simulated using the different types of edge and cloud servers listed in Table 4. Servers 1 and 2 originate from the Intelligent Distributed Computing and Systems (INDUCE) Research Laboratory, College of Information Technology, United Arab Emirates University. The specifications of the remaining servers, 3 – 6 are taken from the SPEC Power benchmark such that they belong to the same family of servers in our laboratory but with different capabilities. We implemented the network using MATLAB 2020a.

Table 4
Specifications of the servers used in the experiments.

Server	Location	Specification	Memory
1	Edge	AMD Opteron 252, 2.59 GHz, 2-Cores	2 GB
2	Cloud	Intel Xeon, 2.80 GHz, 2-Cores	4 GB
3	Edge	AMD Opteron 6276, 2.30 GHz, 16-Cores [45]	32 GB
4	Cloud	Intel Xeon E3-1204L v5, 2.10 GHz, -Cores [46]	16 GB
5	Cloud	Intel Xeon E-2176G, 3.7 GHz, 6-Cores [47]	16 GB
6	Cloud	AMD Opteron 6238, 2.60 GHz, 12-Core [48]	64 GB

Table 5
Network and application characteristics used in the experiments.

Parameter	Value(s)
Number of requests ($\mathcal{I}$)	20, 25, 30, 35, 40, 45, 50
Vehicle – RSU bandwidth (Gbps) $(\omega_{v_h,e_j},\ \forall z \in \mathcal{Z},\ \forall h \in \mathcal{H})$	1
RSU – cloud bandwidth (Gbps) $(\omega_{e_j,c_k},\ \forall z \in \mathcal{Z},\ \forall h \in \mathcal{H})$	U(1, 2)
Time required for data swapping operation (seconds) $(\xi_{s_z}, \forall s_z \in \{e_j, c_k\})$	0.05
Server's CPU utilization threshold $(\varphi^{max}_{s_z}, \forall s_z \in \{e_j, c_k\})$	90
Requests' CPU utilization (%) $(\varphi_{r_i},\ \forall i \in \mathcal{I})$	N(20, 5)
Requests' length (Million Instructions) $(\psi_{r_i},\ \forall i \in \mathcal{I})$	9000 – 15000 [23], [44]
Requests' size (KB) $(\sigma_{r_i},\ \forall i \in \mathcal{I})$	1000 – 5000 [23]
Requests' latency requirements (seconds) $(L^{max}_{r_i},\ \forall i \in \mathcal{I})$	0.1, 0.3, 0.5, 0.7, 0.9, 1.1
Requests' processing time requirements (seconds) $(P^{max}_{r_i},\ \forall i \in \mathcal{I})$	0.9, 1.1, 1.3, 1.5, 1.7, 1.9
Requests' deadline requirements (seconds) $(D^{max}_{r_i},\ \forall i \in \mathcal{I})$	1, 1.2, 1.4, 1.6, 1.8, 2

U denotes uniform distribution; N denotes a standard normal distribution.

Table 6
Genetic algorithm parameters used for convergence analysis.

Parameter	Value(s)
Crossover rate (λ_c)	0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95
Mutation rate (λ_m)	0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1
Population size (P_{size})	$2 \times requests$, $4 \times requests$, $6 \times requests$, $8 \times requests$, $10 \times requests$
Termination condition	1000 iterations (generations)

In our simulated network, we use the Vehicle-Crowd Interaction (VCI) – DUT dataset [43] for vehicles' positioning. In particular, we used the x_est and y_est columns of the dataset for the source and destination locations of vehicles in our experiments. Regarding the characteristics of the vehicular requests, we used three different ITS applications; facial recognition for autonomous driving, augmented reality, and infotainment [23], [44]. The network and application characteristics used in the experiments are listed in Table 5. Table 6 shows the values used for convergence analysis of the proposed QoS-SLA-AGA.

6.2. Experiments

This section explains the experiments performed for convergence analysis of QoS-SLA-AGA and compares its performance with baseline approaches in terms of total execution time (seconds) and the number of requests violating SLA constraints.

To analyze the convergence of the proposed algorithm we executed the algorithm with different values of λ_c, λ_m, and P_{size} (listed in Table 6).

1. We first run the algorithm with varying values of λ_c and keep the values of λ_m and P_{size} constant at 0.01 and $2 \times requests$ respectively.
2. We then select the value of λ_c that has the fastest convergence as the optimal crossover rate.
3. Next, we run the proposed algorithm with varying values of λ_m and the values of λ_c and P_size constant at optimal crossover rate and $2 \times requests$ respectively.
4. We select the value of λ_m resulting in the fastest convergence as the optimal mutation rate.
5. Lastly, we vary the values of P_{size} with λ_c and λ_m constant at their optimal values.
6. We then select the value with the least convergence time as the optimal value for P_{size}.

We evaluate the offloading performance of the proposed algorithm with varying values of latency, processing time, deadline requirements, and the number of requests (Table 5). We vary one of those four parameters while keeping the remaining three constants at their corresponding minimum values. The value for deadline requirement is varied while varying the latency and processing time requirements because the deadline is the summation of latency and processing times. For each run, we use the optimal values of λ_c, λ_m, and P_{size}. We measure the algorithm's performance in terms of the total execution time of the requests and the number of requests violating SLA requirements. For latency, processing time, and deadline violations, we calculate the number of requests for which the value of communication time, processing time, and total execution time is greater than the requests' requirements. To determine the number of requests violating CPU requirements, we consider the number of requests scheduled on a server where the total CPU utilization of all requests scheduled on that server exceeds the server's CPU utilization threshold.

To demonstrate the performance of the proposed QoS-SLA-AGA, we compare it with the following baseline approaches:

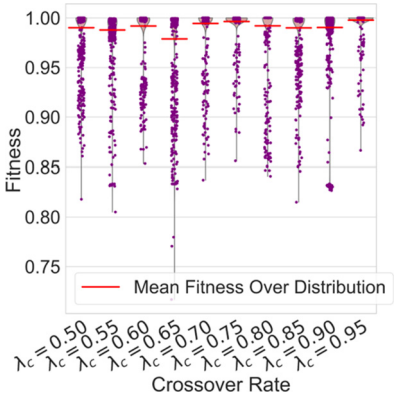


Fig. 5. Fitness score distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus crossover rate λ_c.

Table 7
Optimal values of genetic algorithm parameters.

Parameter	Value(s)
Crossover rate (λ_c)	0.95
Mutation rate (λ_m)	0.01
Population size (P_{size})	$2 \times requests$

1) **QoS-Aware Genetic Algorithm (QoS-GA):** An offloading scheme using a genetic algorithm whose objective is to minimize the total execution time of all the requests without considering the SLA constraints.
2) **QoS-Aware Particle Swarm Optimization (QoS-PSO):** An offloading scheme using a PSO algorithm whose objective is to minimize the total execution time of all the requests without considering the SLA constraints. PSO is selected as a baseline approach as it is a widely used meta-heuristic approach for offloading in literature [14], [15], [27].
3) **Random Offloading:** An offloading scheme where each request is randomly scheduled at the edge or cloud server without considering the QoS and SLA.
4) **All Edge Computing (AEC):** An approach where each request is executed by the edge server to which it has been submitted.
5) **All Cloud Computing (ACC):** An approach where each request is executed by one of the cloud servers in a way that minimizes the total execution time of all the requests while considering the SLA constraints.

We repeat the experiments for QoS-GA, QoS-PSO, random offloading, AEC, and ACC with varying SLA requirements and the number of requests.

6.3. Experimental results analysis

This section presents the analysis of the results obtained from our experiments. In particular, we analyze the convergence results of our proposed algorithm and compare our algorithm with baseline approaches.

6.3.1. Convergence analysis

Fig. 5 shows the convergence of QoS-SLA-AGA in terms of penalized fitness score distribution over iterations with varying values of λ_c. As shown, for all values of λ_c, the distributions over iterations are left-skewed. However, $\lambda_c = 0.95$, has the highest mean fitness score of 0.997. Consequently, value $\lambda_c = 0.95$ converges the algorithm in the number of iterations. This is because a higher crossover rate diversifies the population by selecting more offloading solutions to perform the crossover operation. On the other hand, the offloading solutions are not as diverse when the crossover rate is low. Fig. 6 shows the distribution of the total execution time of the requests over iterations with varying values of λ_c. As shown in the figure, $\lambda_c = 0.85$ which results in the fastest convergence with the shortest total execution time. Although, $\lambda_c = 0.85$ optimizes the total execution time, it does not provide the best convergence of the algorithm in terms of fitness (Fig. 5). This is because the fitness score in Fig. 5 considers both the optimization of total execution time and SLA violations, whereas Fig. 6 only considers the optimization of total execution time. Consequently, $\lambda_c = 0.85$ optimizes the time but violates SLA constraints. On the other hand, $\lambda_c = 0.95$ converges to an offloading solution with a minimum execution time that respects SLA constraints. Consequently, we use $\lambda_c = 0.95$ in the remaining experiments.

Fig. 7 shows the fitness convergence of QoS-SLA-AGA with varying values of λ_m. As depicted in the figure, only $\lambda_m = 0.01$ converges the algorithm to an optimal fitness value of 1. This is because a higher mutation rate hinders the convergence as fitter offloading solutions are lost. Fig. 8 depicts the distribution of total execution time over iterations with varying values of λ_m. As shown in the figure, only $\lambda_m = 0.01$ converges the algorithm to the minimum execution time. Consequently, we use $\lambda_m = 0.01$ in the offloading experiments.

Fig. 9 shows the fitness convergence of QoS-SLA-AGA with a varying value of P_{size}. As shown, the algorithm converges to the optimal fitness score of 1 with the highest mean when the population size is set to twice the number of requests. As shown in Fig. 10, the total execution time of the requests converges to the minimum quickly when $P_{size} = 2 \times requests$. Consequently, we use $P_{size} = 2 \times requests$ in the experiments. Table 7 shows the optimal genetic parameters used to evaluate the performance of the proposed QoS-SLA-AGA and baseline QoS-GA algorithms.

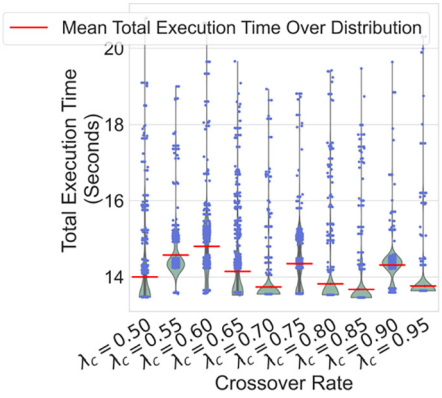


Fig. 6. Total execution time distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus crossover rate λ_c.

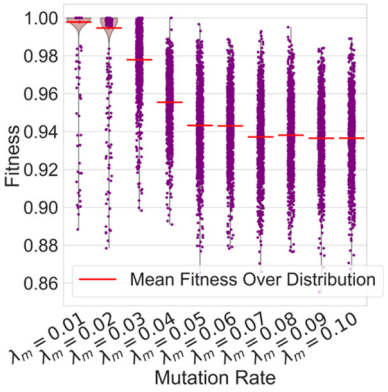


Fig. 7. Fitness score distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus mutation rate λ_m.

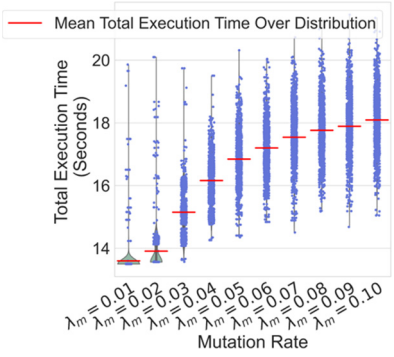


Fig. 8. Total execution time distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus mutation rate λ_m.

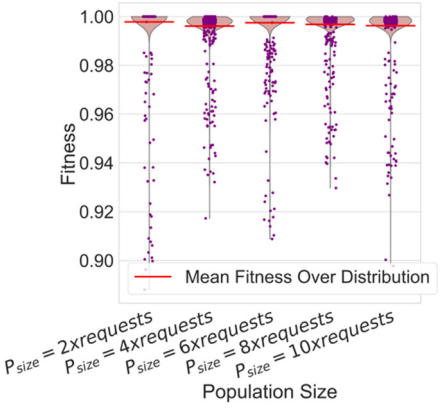


Fig. 9. Fitness score distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus population size P_{size}.

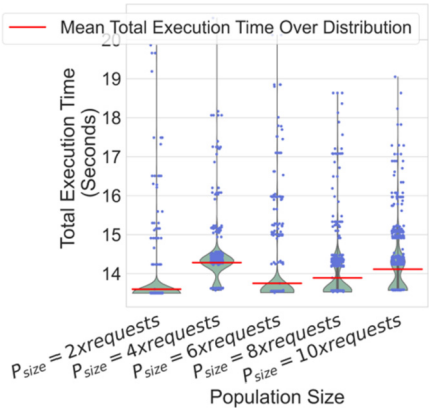


Fig. 10. Total execution time distribution over iterations for the requests' offloading solution using QoS-SLA-AGA versus population size P_{size}.

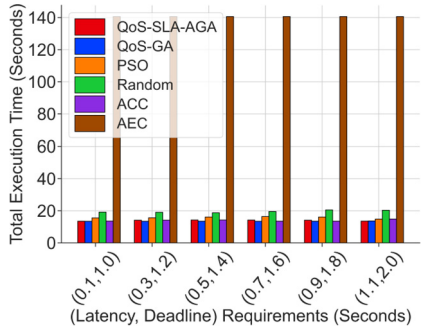


Fig. 11. Total execution time using QoS-SLA-AGA and baseline approaches versus latency requirements.

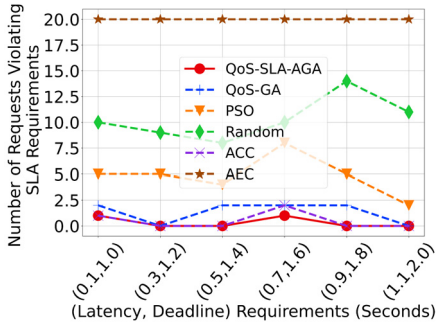


Fig. 12. Number of requests violating SLA requirements using QoS-SLA-AGA and baseline approaches versus latency requirements.

6.3.2. Comparative performance analysis

Figs. 11 and 12 show the total execution time and the number of requests violating SLA requirements respectively, for QoS-SLA-AGA, QoS-GA, PSO, random offloading, ACC, and AEC solutions with increasing latency requirements. As shown in Fig. 11, AEC has the longest execution time, as all the requests are processed on edge servers having lower computing capabilities compared to the cloud servers. The proposed QoS-SLA-AGA outperforms PSO, random, and AEC approaches providing lower execution time. While QoS-GA and ACC have better performance than the proposed QoS-SLA-AGA (Fig. 11), they have more SLA violations (Fig. 12) compared to the proposed algorithm. This is because QoS-GA minimizes the total execution time without considering the SLA constraints and ACC process everything on cloud servers, without considering edges, leading to higher communication time. On the other hand, QoS-SLA-AGA uses both edge and cloud servers to process requests in a way that minimizes the total execution time while considering the constraints. In summary, the average total execution times with increasing latency requirements for QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 13.97 seconds, 13.52 seconds, 15.77 seconds, 19.57 seconds, 13.99 seconds, and 140.59 seconds respectively. The average number of requests violating SLA constraints using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 0.33, 1.33, 4.83, 10.33, 0.5, and 20 respectively. On average, with increasing latency requirements, 1.66%, 6.66%, 24.16%, 51.66%, 2.5%, and 100% requests violate SLA constraints using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC approaches respectively.

Figs. 13 and 14 show the total execution time and the number of requests violating SLAs, respectively, for QoS-SLA-AGA, QoS-GA, PSO, random offloading, ACC, and AEC algorithms with increasing processing time requirements. As shown in Fig. 13, the AEC approach has the longest execution time with increasing processing time requirements. Compared to QoS-GA, QoS-SLA-AGA results in a higher total execution time. This is because of the SLA constraints consideration in the proposed algorithm in addition to the objective of minimizing the total execution time. As shown in Fig. 14, QoS-SLA-AGA has the least SLA violations. However, QoS-GA violates more requests compared to QoS-SLA-AGA. AEC violates the maximum number of requests as the edge servers become bottlenecks leading to higher processing time.

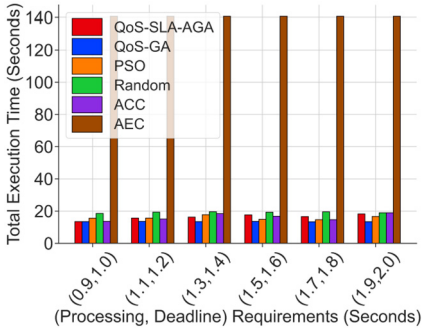


Fig. 13. Total execution time using QoS-SLA-AGA and baseline approaches versus processing time requirements.

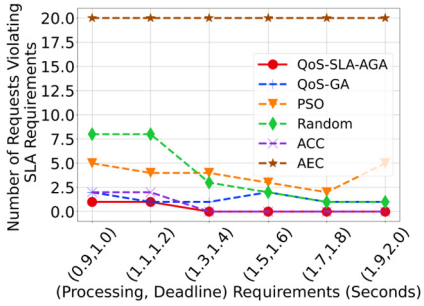


Fig. 14. Number of requests violating SLA requirements using QoS-SLA-AGA and baseline approaches versus processing time requirements.

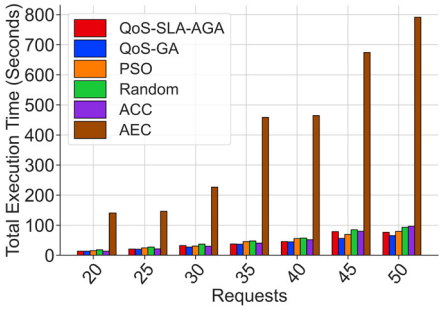


Fig. 15. Total execution time using QoS-SLA-AGA and baseline approaches versus the number of requests.

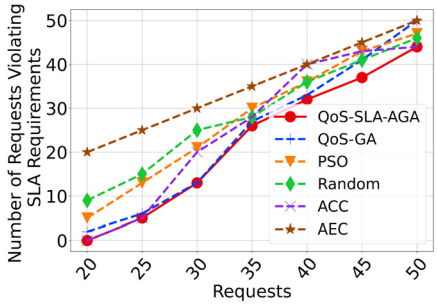


Fig. 16. Number of requests violating SLA requirements using QoS-SLA-AGA and baseline approaches versus the number of requests.

In summary, the average total execution times with increasing processing time requirements for QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 16.33 seconds, 13.49 seconds, 15.88 seconds, 19.21 seconds, 16.25 seconds, and 140.59 seconds respectively. The average number of requests violating SLA constraints using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 0.33, 1.33, 3.83, 3.83, 0.66, and 20 respectively. On average, with increasing processing requirements, 1.66%, 6.66%, 19.16%, 19.16%, 3.33%, and 100% of requests violate SLA constraints using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC approaches respectively.

Figs. 15 and 16 show the total execution time and the number of requests violating SLAs, respectively, for QoS-SLA-AGA, QoS-GA, PSO, random offloading, ACC, and ACC algorithms with an increasing number of requests. As shown in Fig. 15, AEC has the longest execution time with an increasing number of requests. QoS-GA outperforms the proposed approach with an increasing number of requests, whereas PSO outperforms the proposed algorithm for 45 requests. However, the proposed algorithm has the least SLA violations (Fig. 16). As shown in Fig. 16, all algorithms violate the SLA requirements with AEC having the highest violations. The SLA violations increase with the number of requests for all the algorithms. This is because the processing power of the servers is divided among requests with increasing requests

while keeping the number of servers constant. Consequently, the processing time of requests increases leading to increased total execution time. In summary, the average total execution times with an increasing number of requests using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 43.63 seconds, 37.75 seconds, 45.85 seconds, 52.22 seconds, 47.68 seconds, and 414.68 seconds respectively. The average number of requests violating SLA requirements using QoS-SLA-AGA, QoS-GA, PSO, random, ACC, and AEC algorithms are 22.2, 24, 27.85, 28.57, 25.71, and 35 respectively. On average, with increasing requests, 55.40%, 59.52%, 73.18%, 77.34%, 64.31%, and 100% of requests violate SLA constraints using QoS-SLA-AGA, QoS-AGA, PSO, random, ACC, and AEC approaches respectively.

7. Conclusion

Computation offloading is essential in an integrated edge-cloud system for IoV to enhance the QoS and respect the SLA requirements of both compute-intensive and time-critical applications. In this paper, we propose QoS-SLA-AGA to offload vehicular applications' requests on an edge/cloud server such that the total execution time of the requests is minimized. Furthermore, our proposed optimization algorithm is constrained by the requests' SLA requirements in terms of latency, processing time, deadline, CPU, and memory. The proposed algorithm considers the overlapping of requests execution in the offloading decision. To the best of our knowledge, we are the first to propose a QoS-SLA-aware offloading algorithm using AGA in IoV that considers the overlapping of multi-request execution and dynamic speed of the vehicle for execution time minimization while adhering to the performance and resource SLA constraints. Numerical experiments and comparative analysis revealed that the proposed algorithm outperforms the random offloading, PSO, ACC, and AEC approaches in total execution time. In the context of SLA constraints, the proposed algorithm outperforms all the baseline approaches. In future research, we propose to investigate QoS-SLA-aware priority-based partial offloading solutions with inter-dependency among requests while making the offloading decision. Furthermore, we aim to reduce the time for obtaining the optimal solution (i.e., convergence time) and to compare a large spectrum of offloading algorithms including the ones based on deep reinforcement learning.

Funding

This research was funded by the National Water and Energy Center of the United Arab Emirates University (Grant 12R126).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Leila Ismail reports financial support was provided by United Arab Emirates University.

Data availability

Data will be made available on request.

References

[1] S. Kumar, J. Singh, Internet of vehicles over Vanets: smart and secure communication using IoT, Scalable Comp. Pract. Exp. 21 (3) (2020) 425–440.
[2] K. Guan, et al., 5-GHz obstructed vehicle-to-vehicle channel characterization for Internet of intelligent vehicles, IEEE Int. Things J. 6 (1) (2018) 100–110.
[3] J. Contreras-Castillo, S. Zeadally, J.A. Guerrero-Ibanez, Internet of vehicles: architecture, protocols, and security, IEEE Int. Things J. 5 (5) (2017) 3701–3709.
[4] J.A. Guerrero-Ibanez, S. Zeadally, J. Contreras-Castillo, Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and Internet of things technologies, IEEE Wirel. Commun. 22 (6) (2015) 122–128.
[5] C.-C. Lin, D.-J. Deng, C.-C. Yao, Resource allocation in vehicular cloud computing systems with heterogeneous vehicles and roadside units, IEEE Int. Things J. 5 (5) (2017) 3692–3700.
[6] L. Ismail, R. Barua, Implementation and performance evaluation of a distributed conjugate gradient method in a cloud computing environment, Softw. Pract. Exp. (2012).
[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] 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, 2020, pp. 161–176.
[9] Q.-V. Pham, H.T. Nguyen, Z. Han, W.-J. Hwang, Coalitional games for computation offloading in NOMA-enabled multi-access edge computing, IEEE Trans. Veh. Technol. 69 (2) (2019) 1982–1993.
[10] X. Xu, X. Zhang, X. Liu, J. Jiang, L. Qi, M.Z.A. Bhuiyan, Adaptive computation offloading with edge for 5G-envisioned Internet of connected vehicles, IEEE Trans. Intell. Transp. Syst. 22 (8) (2021) 5213–5222.
[11] M. Khayyat, I.A. Elgendy, A. Muthanna, A.S. Alshahrani, S. Alharbi, A. Koucheryavy, Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks, IEEE Access 8 (2020) 137052–137062.
[12] M. Ibrar, et al., ARTNet: AI-based resource allocation and task offloading in a reconfigurable Internet of vehicular networks, IEEE Trans. Netw. Sci. Eng. 9 (1) (Jan. 2022) 67–77, https://doi.org/10.1109/TNSE.2020.3047454.
[13] X. Xu, et al., A computation offloading method over big data for IoT-enabled cloud-edge computing, Future Gener. Comput. Syst. 95 (2019) 522–533.
[14] S.A. Zakaryia, S.A. Ahmed, M.K. Hussein, Evolutionary offloading in an edge environment, Egypt. Inform. J. 22 (3) (Sep. 2021) 257–267, https://doi.org/10.1016/j.eij.2020.09.003.
[15] Z. Zhao, et al., A novel framework of three-hierarchical offloading optimization for MEC in industrial IoT networks, IEEE Trans. Ind. Inform. 16 (8) (Aug. 2020) 5424–5434, https://doi.org/10.1109/TII.2019.2949348.
[16] I. Sorkhoh, D. Ebrahimi, R. Atallah, C. Assi, Workload scheduling in vehicular networks with edge cloud capabilities, IEEE Trans. Veh. Technol. 68 (9) (2019) 8472–8486.
[17] H. Yuan, M. Zhou, Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems, IEEE Trans. Autom. Sci. Eng. 18 (3) (2021) 1277–1287.
[18] H. Zhou, K. Jiang, X. Liu, X. Li, V.C. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile edge computing, IEEE Int. Things J. (2021).
[19] Z. Xu, W. Liang, M. Jia, M. Huang, G. Mao, Task offloading with network function requirements in a mobile edge-cloud network, IEEE Trans. Mob. Comput. 18 (11) (2018) 2672–2685.
[20] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, M. Ylianttila, Cloud and edge computation offloading for latency limited services, IEEE Access 9 (2021) 55764–55776.
[21] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, M. Xu, EEDTO: an energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing, IEEE Int. Things J. 8 (4) (2020) 2163–2176.
[22] T. Wang, et al., An intelligent dynamic offloading from cloud to edge for smart IoT systems with big data, IEEE Trans. Netw. Sci. Eng. 7 (4) (2020) 2598–2607.

[23] J. Almutairi, M. Aldossary, A novel approach for IoT tasks offloading in edge-cloud environments, J. Cloud Comput. 10 (1) (2021) 1–19.
[24] Y. Wang, et al., A game-based computation offloading method in vehicular multiaccess edge computing networks, IEEE Int. Things J. 7 (6) (2020) 4987–4996.
[25] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Trans. Veh. Technol. 68 (8) (2019) 7944–7956.
[26] Y. Dai, D. Xu, S. Maharjan, Y. Zhang, Joint load balancing and offloading in vehicular edge computing and networks, IEEE Int. Things J. 6 (3) (2018) 4377–4387.
[27] C. Zhu, et al., Folo: latency and quality optimized task allocation in vehicular fog computing, IEEE Int. Things J. 6 (3) (2018) 4150–4161.
[28] G. Peng, H. Wu, H. Wu, K. Wolter, Constrained multi-objective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing, IEEE Int. Things J. 8 (17) (2021) 13723–13736.
[29] H. Liao, et al., Learning-based intent-aware task offloading for air-ground integrated vehicular edge computing, IEEE Trans. Intell. Transp. Syst. 22 (8) (Aug. 2021) 5127–5139, https://doi.org/10.1109/TITS.2020.3027437.
[30] L. Ismail, H. Materwala, ESCOVE: energy-SLA-aware edge-cloud computation offloading in vehicular networks, Sensors 21 (15) (2021) 5233.
[31] G. Zhang, T.Q.S. Quek, M. Kountouris, A. Huang, H. Shan, Fundamentals of heterogeneous backhaul design—analysis and optimization, IEEE Trans. Commun. 64 (2) (2016) 876–889.
[32] L. Hu, et al., Optimal route algorithm considering traffic light and energy consumption, IEEE Access 6 (2018) 59695–59704.
[33] L. Ismail, H. Materwala, EATSVM: energy-aware task scheduling on cloud virtual machines, Proc. Comput. Sci. 135 (2018) 248–258, https://doi.org/10.1016/j.procs.2018.08.172.
[34] C. Chekuri, S. Khanna, A polynomial time approximation scheme for the multiple knapsack problem, SIAM J. Comput. 35 (3) (2005) 713–728.
[35] D. Pisinger, Where are the hard knapsack problems?, Comput. Oper. Res. 32 (9) (2005) 2271–2284.
[36] H. Materwala, L. Ismail, R.M. Shubair, R. Buyya, Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks, Future Gener. Comput. Syst. 135 (2022) 205–222, https://doi.org/10.1016/j.future.2022.04.009.
[37] J.H. Holland, Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, 1992.
[38] B. Tessema, G.G. Yen, An adaptive penalty formulation for constrained evolutionary optimization, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 39 (3) (2009) 565–578.
[39] 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.
[40] J.A. Lozano, P. Larrañaga, M. Graña, F.X. Albizuri, Genetic algorithms: bridging the convergence gap, Theor. Comput. Sci. 229 (1–2) (Nov. 1999) 11–22, https://doi.org/10.1016/S0304-3975(99)00090-0.
[41] A.E. Eiben, E.H.L. Aarts, K.M. Van Hee, Global convergence of genetic algorithms: a Markov chain analysis, https://doi.org/10.1007/BFb0029725, 1991, pp. 3–12.
[42] J. He, L. Kang, On the convergence rates of genetic algorithms, Theor. Comput. Sci. 229 (1–2) (Nov. 1999) 23–39, https://doi.org/10.1016/S0304-3975(99)00091-2.
[43] D. Yang, L. Li, K. Redmill, Ü. Özgüner, Top-view trajectories: a pedestrian dataset of vehicle-crowd interaction from controlled experiments and crowded campus, in: 2019 IEEE Intelligent Vehicles Symposium (*IV*), 2019, pp. 899–904.
[44] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.
[45] Edge Server 1, https://www.spec.org/power_ssj2008/results/res2012q1/power_ssj2008-20120306-00437.html, 2012. (Accessed 4 January 2022).
[46] SPECpower, Cloud Server 1, https://www.spec.org/power_ssj2008/results/res2016q1/power_ssj2008-20151214-00707.html, 2016. (Accessed 4 January 2022).
[47] SPECpower, Cloud Server 2, https://www.spec.org/power_ssj2008/results/res2019q2/power_ssj2008-20190507-00964.html, 2018. (Accessed 4 January 2022).
[48] Cloud Server 3, https://www.spec.org/power_ssj2008/results/res2012q1/power_ssj2008-20120213-00420.html, 2012. (Accessed 4 January 2022).