Distributed Clustering-Based Cooperative Vehicular Edge Computing for Real-Time Offloading Requests

Junhua Wang, Kun Zhu, *Member, IEEE*, Bing Chen, and Zhu Han, *Fellow, IEEE*

***Abstract*—Mobile vehicles have been considered as potential edge servers to provide computation resources for the emerging Intelligent Transportation System (ITS) applications. However, how to fully utilize the mobile computation resources to satisfy the real-time arrived computation requests has not been explored yet. This work will address the critical challenges of limited computation resources, stringent computation delay and unknown requirement statistics of real-time tasks in realistic vehicular edge computing scenarios. Specifically, we design a distributed clustering strategy to classify vehicles into multiple cooperative edge servers according to the available computation resources, effective connection time and the distribution of tasks' expected deadlines. Then, a 'Less than or Equal to' Generalized Assignment Problem (i.e., LEGAP) is formulated to maximize the system service revenue, and on this basis, we propose an offline Bound-and-Bound based Optimal (BBO) algorithm to make periodical scheduling with a global view of tasks' requirement statistics. The quick branching is conducted by following a greedy solution and the upper bound at each branch is derived by solving a multiple-choice knapsack problem. In addition, we present an online heuristic algorithm which makes real-time offloading decision with the guarantees that the resource capacities of all the computing servers are never exceeded with new tasks arriving. Through comparing with the other four online algorithms, the BBO algorithm achieves the highest service revenues by offloading tasks with shorter delays, and the online heuristic algorithm has the best performance in improving the service ratios.**

***Index Terms*—Cooperative vehicular edge computing, distributed clustering, online scheduling, bound-and-bound algorithm.**

I. Introduction

IN THE next-generation Intelligent Transportation Systems (ITSs), billions of traffic devices (i.e., roadside cameras, traffic lights and mobile vehicles) will be connected for the information exchange and coordination management, which requires numerous edge servers to provide quick responses and high-quality services for emerging massive computation-intensive requests [1], [2].

On one hand, considering the ultra-reliable low-latency requirements of on-board applications, some researches aimed at providing vehicle users with adequate computation resources from the edge network [3]–[5]. New mobile edge computing (MEC) technologies have been proposed in dealing with the frequent service handover between multiple edge servers [6]–[8]. For example, in ultra-dense network, the user-centric energy-aware mobility management (EMM) scheme was proposed for mobile users to select edge servers in the handover process [9]. Task replication technologies can also be applied to improve the computation reliability in dynamic vehicular edge computing environment [10], [11].

On the other hand, since future vehicles are expected to be equipped with more powerful on-board computers [12], they may have extra computation resources, especially when waiting/parking at certain places or driving during the off-peak time [13], [14]. Thus, some researches considered to support the intelligent traffic applications via forming a vehicular edge/fog computing network [15], [16]. In [17], the authors presented the vehicular fog computing system (VFCS) to utilize the public service vehicles (i.e., buses) as fog servers. An application-aware offloading policy was proposed to optimize the computation resource allocation and the long-term expected reward of the VFCS. In [18], considering the distinct features of mobile fog layer (i.e., formed by moving vehicles), fixed fog layer and the cloud server in terms of communication rates and computation capabilities, the multi-task computation offloading problem was formulated as a multi-dimensional multiple knapsack problem (MMKP) and solved by the optimal algorithm. In addition, the vehicles can be employed as both the resource providers and resource subscribers in emerging ITS applications [19], [20]. For example, in order to minimize the response time of citywide events reported by vehicles, the authors constructed a three-layer vehicular fog computing (VFC) model with both moving and parked vehicles as fog nodes to enable distributed traffic management [21]. In [22], the authors investigated on the vehicular edge computing in which the collaborative computing was enabled via V2V (i.e., vehicle-to-vehicle) communications, and then, a deep reinforcement learning (DRL) based collaborative data scheduling scheme was developed to minimize the system-wide data processing cost.

Manuscript received February 18, 2021; revised July 14, 2021 and September 29, 2021; accepted October 17, 2021. Date of publication October 21, 2021; date of current version January 20, 2022. This work was supported in part by the National Natural Science Foundation of China under Grants 62002166 and 62071230, in part by the Natural Science Foundation of Jiangsu Province under Grants BK20200419 and BK20211567, and in part by the China Postdoctoral Science Foundation under Grant 2020M671483. The review of this article was coordinated by Prof. Kyunghan Lee. *(Corresponding author: Kun Zhu.)*

Junhua Wang, Kun Zhu, and Bing Chen are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China (e-mail: jhua1207@nuaa.edu.cn; zhukun@nuaa.edu.cn; cb_china@nuaa.edu.cn).

Zhu Han is with the University of Houston, Houston, TX 77004 USA, and also with the Department of Computer Science and Engineering, Kyung Hee University, Seoul 130-701, South Korea (e-mail: zhan2@uh.edu).

Digital Object Identifier 10.1109/TVT.2021.3122001

However, it remains unexplored how to make full use of the mobile computation resources on the road for meeting the real-time arrived offloading requests. First, for computation-intensive tasks, it's necessary to consider how to employ multiple vehicles as a powerful edge server for cooperative computation. The decentralized management of mobile computation resources has not been researched in current literatures. Second, in dynamic traffic scenario where moving vehicles have different connection time and resource capacities, how to assign subtasks to multiple vehicles and coordinate them to complete a larger computation task is a critical problem. Third, in the real-time offloading environment, the required information of communication/computation resources of the following tasks are unknown before their arrivals. The current offloading decision will affect the future offloading decision and even the global scheduling performance. Therefore, it is difficult to determine which one of the periodical scheduling and online scheduling solutions can achieve the best performance.

In this work, we focus on a hybrid computation offloading scenario with mobile edge servers (i.e., vehicles), static edge servers and the cloud server as resource providers. The real-time computation tasks generated by end users (i.e., moving vehicles and roadside cameras) may require to be offloaded to the heterogenous computing servers for fast processing. Since the end users have very limited information about the available computation resources from surroundings, the road-side units (i.e., RSUs) are deployed to collect the request messages and make offloading decisions. In order to fully utilize the mobile computation resources on the road, we propose the distributed clustering-based cooperative vehicular edge computing architecture and the periodical/online offloading algorithms. The main contributions of this paper are outlined as follows:

- We present a cooperative vehicular edge computation offloading architecture, where a distributed clustering mechanism is designed to classify vehicles into multiple cooperative edge servers according to the available computation resources, the effective connection time and the distribution of tasks' expected deadlines. The cluster head that is chosen based on its contribution value coordinates the cluster members to complete tasks in parallel.
- We formulate a realistic computation offloading problem (i.e., LEGAP) which can be transformed into an offline periodical scheduling problem and an online scheduling problem by adjusting the waiting time of tasks. In the online scheduling scenario, the requirement statistics (i.e., demanding communication/computation resources and expected deadlines) of the following tasks are unknown before their arrivals.
- We propose the offline BBO algorithm which makes the optimal offloading decision periodically with a global view of all the tasks' requirement information. Different from the traditional branch-and-bound algorithm, the upper bound at each branch node is computed by transforming the allocation subproblem into a multiple choice knapsack problem (i.e., MCKP), and the branching process is conducted by following a greedy solution.
- Further, we propose a low-complexity online heuristic algorithm which makes the real-time offloading decision at the arrival of each computation task. The resource capacities of all the computing servers are never exceeded during the entire scheduling period.
- We compare the performances of the proposed algorithms with the online DRL-based algorithm, online random algorithm, online Revenue_First algorithm and online R2C_First algorithm. Comprehensive studies demonstrate the advantages of the offline BBO algorithm on the system revenue, and the advantages of the online heuristic algorithm on improving the service ratio.

The reminder of this paper is organized as follows. Section II reviews the related literatures. Section III presents the system model. In Section IV, we propose the distributed clustering approach and formulate an LEGAP. In Section V, the BBO and online heuristic algorithms are proposed. The simulation results are given in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

A. MEC for Vehicle Users

A considerable number of researches have been conducted on the optimization of edge computation offloading decisions, especially for reducing the computation delay of tasks and the energy consumption of edge servers under different network models and resource constraints [23]–[29]. The development of ITS applications promotes the exploration in using edge computing paradigms to serve for intensive computation requirements of mobile vehicles [30]–[33]. L. Zhao *et al.* [31] proposed a UAV (i.e., unmanned air vehicle) assisted Vehicular computation Cost Optimization (UVCO) algorithm to solve the multi-players computation offloading sequential game, where the UAV can compute tasks for vehicles or act as a relay between edge servers and vehicles. H. Liao *et al.* [32] presented a air-ground integrated vehicular edge computing (AGI-VEC) framework with UAVs and base stations as edge servers. The proposed learning-based intent-aware upper confidence bound (IUCB) algorithm enabled a vehicle user to learn the long-term optimal task offloading strategy under the ultra-reliable low-latency communication (URLLC) constraints with information uncertainty. H. Wu *et al.* [33] considered the joint optimization of UAV content placement, content delivery and flying trajectory in the process of serving vehicular demands. They proposed a learning-based scheme to maximize the overall network throughput in real-time offloading environments. Specifically, the optimal scheduling solution obtained from an offline joint caching and trajectory optimization (JCTO) algorithm was used to train the agent via better guidance.

B. MEC With Vehicle Servers

Recent researches have considered to utilize the potential computing resources of mobile vehicles to serve nearby traffic requests [34]–[38]. Z. Zhou *et al.* [34] proposed a contract-based server recruitment mechanism to motivate vehicular fog servers

(VFSs) as resource providers, and formulated a two-dimensional matching between user vehicles and VFSs. Then, they proposed the pricing-based matching algorithm and the multi-armed bandit (MAB) based mechanism with the known and unknown global offloading information, respectively. L. Pu *et al.* [35] presented a hybrid edge computing framework named Chimera in vehicular crowd sensing applications, and designed a Lyapunov optimization based task scheduling algorithm which enabled vehicles to process data via choosing the local computing, the V2V-based computing or the edge cloud based computing strategies. J. Feng *et al.* [36] proposed an Autonomous Vehicular Edge (AVE) architecture to support the vehicular computation offloading, which supported the autonomous organization of the vehicular cloud via a decentralized procedure. With the caching mechanism applied, an Ant Colony Optimization (ACO) based scheduling algorithm was also proposed to solve the task allocation problem. J. Du *et al.* [37] formulated a dual-side optimization problem to minimize the resource cost of vehicular terminals (VTs) and MEC servers in cognitive vehicular networks. Based on Lyapunov optimization, a dynamic joint task offloading and resource allocation algorithm was proposed to optimize the offloading decision and local CPU frequency on the VT side, and the radio resource allocation and server provisioning on the server side. Y. Sun *et al.* [38] introduced the task replication to the vehicular edge computing system, where the replicas of a task were offloaded to multiple vehicles for providing reliable computing services. They obtained the approximated closed-form solution for the optimal number of task replicas, and on this basis, designed a distributed learning-based task replication algorithm (LTRA) to minimize the offloading delay in the dynamic VEC system.

C. Cooperative Vehicular Edge Computing

The computation capability of a single vehicle may not be adequate for completing a large computation task. Recent researches have considered to coordinate multiple mobile vehicles for completing tasks in a distributed way [39]–[42]. H. Sami *et al.* [39] proposed a framework called Vehicular-OBUs-As-On-Demand-Fogs, which adopted the containerization technologies, Docker, to deploy micro-services on OBUs. They formulated the resource selection and micro-services placement problem on clusters of vehicles and solved it using an Evolutionary Memetic Algorithm (EMA). X. Han *et al.* [40] derived the coupling reliability model for cooperative communication and computation system. Considering the amount of processing data and cut-off time of vehicle applications, they proposed the coupling-oriented reliability calculation for vehicle collaboration computing based on dynamic programming methods. P. Dai *et al.* [41] formulated a vehicular cooperative computation offloading (CCO) problem by modeling the procedure of coordinated task upload, migration and computation based on queuing theory, and then, proposed a probabilistic computation offloading algorithm (PCO) to minimize the completion delay in an iterative way. F. Sun *et al.* [42] considered the cooperative vehicular computation offloading where the computation mission was divided into multiple tasks with interdependency and executed at different vehicles, and then, proposed a modified genetic algorithm using integer coding to improve the QoS and resource utilization with the consideration of instable computation resources due to vehicular movement.

D. Online Vehicular Edge Computing

In realistic application scenarios, computation tasks are generated randomly and expected to be processed in real time without waiting for the following tasks. Therefore, it's necessary to explore the online scheduling methods which make offloading decisions without knowing the statistics of computation tasks arrived in the following time slots. X. Wang *et al.* [19] proposed an imitation learning based online task scheduling algorithm to minimize the system energy consumption in real-time offloading scenarios, where the expert obtained the optimal scheduling solution with a few samples in the offline phase, and the agent was trained via online manner by following the expert's demonstration with a theoretical acceptable performance gap. R. Zhang *et al.* [20] formulated the vehicular task offloading as a mortal multi-armed bandit problem, where 'playing' an arm at each round was treated as selecting an edge node to perform task offloading. Considering the dynamic network conditions, an online algorithm was designed to enable distributed decision making of edge node selection. Z. Wang *et al.* [43] proposed an online offloading scheduling and resource allocation (OOSRA) algorithm in vehicular edge computing environment. Specifically, the task offloading scheduling was solved via a game-theoretic online algorithm, and then, the resource allocation problem was dealt with an online bin-packing algorithm which can be adapted to dynamic traffic flow and service attributes. S. Huang *et al.* [44] considered the joint optimization of communication and computation with the random device arrivals in MEC environment. They adopted the one-step policy iteration with a well-designed baseline scheduling to obtain a sub-optimal scheduling policy, and then, proposed an online scheduling algorithm integrating reinforcement learning and stochastic gradient descent (SGD) to optimize the power consumption with the consideration of unknown statistics of random device arrivals.

E. Differences With This Work

Different from above researches, to make better use of the mobile computation resources, we propose a distributed clustering approach based on the overall evaluation of the available resource capacities, the effective connection time and the distribution of tasks' expected deadlines. Each cluster acts as a mobile edge server, within which the vehicles can cooperatively complete a large computation task without exchanging intermittent results with other edge servers. On this basis, for scheduling real-time arrived offloading requests, we propose the offline periodically-scheduling optimal algorithm and online heuristic algorithm, and compare the performances with various online algorithms under wide simulation settings.

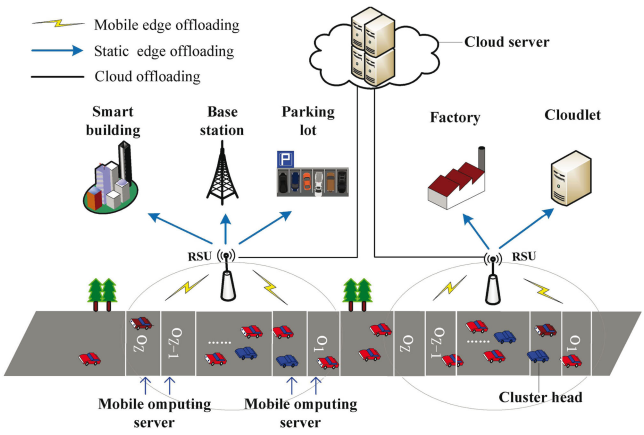

Fig. 1. System model.

III. System Model

We consider a city area equipped with common network infrastructures, as shown in Fig. 1. The fixed network infrastructures such as smart buildings, shopping malls, and parking lots may have redundant communication and computation capabilities during the off-peak time, which can be employed as static edge servers. For example, the parking lot of a large supermarket has available computation resources with the arriving of consumers. In addition, moving vehicles that are passing through the RSU's coverage carry with powerful onboard computers. When vehicles are driving smoothly in safe traffic environments, the computation consumptions of onboard computers are reduced and extra computation capabilities remain unused. Particularly, the congested vehicles have more stable network connectivity to the RSU, which facilitates the computation offloading process. In this paper, we propose to divide multiple geographic zones (i.e., o_1,..., o_Z in Fig. 1) within the RSU's coverage, and select part of vehicles to form a mobile edge server within each zone for cooperative computing. Besides, the remote cloud server has been considered as the traditional computation entity from the past few decades. In the cloud center, large number of computing severs are connected to complete tasks at very high computation frequencies. Compared with the mobile/static edge nodes, the cloud center is considered to have powerful computation capability.

The detailed offloading process is described as follows. Firstly, the roadside users offload request messages that include the required communication/computation resources, the available transmission power, and the expected service delay of the task to the RSU. Then, the RSU makes scheduling according to the received request messages. The scheduling decision contains the user's ID, the server's ID, and the allocated transmission bandwidth and computing frequency. Since the request messages and scheduling messages are of small size, the time and energy consumption in making scheduling decisions can be ignored. Finally, the scheduling decision is returned back to the users, which will offload their tasks by following the scheduling decision. Specifically, the task transmission between the vehicles, and from vehicles to the RSU are conducted via either the V2V interface supported by the dedicated short-range communication (DSRC), or the Cellular-Vehicle-to-Everything (C-V2X) interface. Since static edge servers are near to the RSU and connected via the wired network, the transmission delay between them is ignored. In addition, the cloud server and the RSU can be connected via the fiber optic network, and the transmission delay between them is assumed as a constant [45].

TABLE I
Main Notations

Notation	Description
Z	Number of zones within the RSU's coverage
o_z	The zth zone within the RSU's coverage
h_z	The mobile edge server in zone o_z
e_m	The mth vehicle in the current system
ct_n^m	Connection time of e_m with its nth one-hop neighbor
cp_n^m	Computation rate of the nth one-hop neighbor of e_m
cv_n^m	Contribution value of the nth one-hop neighbor of e_m
dt_m	Dwelling time of e_m within the RSU's coverage
a_j	The jth computation task in the system
G_j	Size of input data of a_j
U_j	Size of computation task of a_j
τ_j	Arrival time of a_j
Δ_j	Waiting time of a_j
D_j	Expected deadline of a_j
s_i	The ith computing server
R_i	Maximum communication capacities of s_i
V_i	Maximum computing capacities of s_i
$v_{i,j}$	Computation frequency of a_j at s_i
$r_{o,j}$	Data rate of a_j from the user to the RSU
$r_{i,j}^0$	Data rate of a_j from the RSU to mobile edge server s_i
$r_{i,j}$	Data rate of a_j from the user to static edge server s_i
σ	Response time via the RSU-cloud link
d_i^{Trasf}	Transferring delay within mobile edge server s_i
$d_{i,j}^f$	Service delay of a_j at static edge server s_i
$d_{i,j}^h$	Service delay of a_j at mobile edge server s_i
$d_{i,j}^c$	Service delay of a_j at the cloud server
dt_i	Effective connection time of s_i to the RSU
$x_{i,j}$	Binary variable indicating whether a_j is offloaded to s_i
$p_{i,j}$	Revenue gained by offloading a_j to s_i
α_1, α_2	Unit price of the communication/computation resources
β	Penalty degree of exceeding the expected deadline
ω	Degree of tolerance

IV. Problem Formulation

In Section IV-A, we propose the distributed clustering approach to manage mobile computation resources by forming multiple cooperative edge servers. In Section IV-B, we propose to evaluate the available resource capacity of each mobile edge server. The computation offloading problem (i.e., LEGAP) is formulated to maximize the service revenues of real-time arrived computation tasks in Section IV-C. For convenience, the main notations are summarized in Table I.

A. Cluster Formation

As suggested in [46], [47], the communication range of a RSU can be divided into a series of adjacent zones via the symmetrical road segment model. On one hand, vehicles in the same zone have approximate relative positions and suffer similar wireless communication conditions within the RSU. On the other hand, multiple nearby vehicles can enhance the computation capability by computing the tasks in parallel. In addition, as elaborated

later, the division of road segment facilitates the distributed clustering within the RSU's coverage. As indicated in [39], vehicles can form a cluster to host micro-services divided from one or more large application services. It requires that the vehicles in the same cluster keep alive connections during the computation process. In this paper, after dividing the communication range of the RSU into multiple geographic zones, we propose to select service vehicles in the same geographic zone as a mobile edge server. The road segment model and the clustering approach are described as follows.

As shown in Fig. 1, there are Z zones in the RSU's coverage. Since the wireless channel quality between the RSU and moving vehicles is highly dependent of the communication distance between them, we simply map the transmission distance to the communication rate by ignoring the influences of shadowing and fading [47]. Each computing server is thus assigned with a distinct transmission rate according to its relative position with the RSU. Note that the 1th and the Zth computing server have the approaching transmission rates due to the symmetric relative physical locations to the RSU. To better utilize the computation capabilities of mobile edge servers, the number of zones (i.e., Z) can be determined according to the computation capacity of mobile vehicles, the size of computation tasks, and the realistic distribution of transmission rates. Since the vehicles in the same zone have different dwelling time, geographic locations and computation capacities, it's not reasonable to employ all the vehicles in the same zone as a cooperative edge server. Therefore, we propose a distributed clustering approach to select service vehicles for forming a powerful edge server. The detailed steps are described as follows:

- In the first phase, we allocate orthogonal resource blocks (RBs) to different zones, so as to avoid the communication interferences between adjacent zones. Vehicles in the middle zones have shorter relative distances to the RSU, and hence, have higher transmission rates than vehicles in the edges of the RSU's coverage.
- In the second phase, with the equipped Global Positioning System (GPS), vehicles are aware of their real-time positions and the relative positions of neighbor vehicles (i.e., vehicles within one-hop communication range). Then, each vehicle broadcasts a message that contains the vehicle's available computation resource, the connection time to each of its neighbor vehicles. The broadcast sequence is decided by mapping the vehicular location sequence. Along with the driving direction, the vehicle in the front of the zone is the first to broadcast, followed by the other vehicles.
- In the third phase, when a vehicle is broadcasting, the other vehicles listen to the broadcasted message and record the available computation resource and the connection time of the broadcasting vehicle. Next, based on the collected information from neighbor vehicles, each vehicle is able to compute its 'overall contribution value' (OCV) based on (2).
- In the forth phase, each vehicle in the zone starts to broadcast its OCV by following the broadcast sequence in the second phase. To reduce the communication overhead, the vehicle broadcasts its OCV only if the OCV is larger than the last vehicle's OCV. The process continues until no other vehicles with higher OCVs appear. Finally, the vehicle with the maximum OCV is successfully selected as the cluster head, and its one-hop neighbors become the cluster members.

As described before, since the cluster members can communicate with the cluster head during the effective connection duration, we consider the tasks can be divided into multiple sub-tasks and completed in parallel by different cluster members [39]. Any two cluster members can exchange the task data and results with each other via the cluster head. Even the length of connection time between cluster members and the cluster head are different, the one that losses connection to the cluster head can transfer its workload to the other cluster members. With this cooperation scheme, the computation resources of all the cluster members are fully utilized. The transferring delay needs to be considered when employing multiple mobile vehicles as a large edge server. In reality, the delay caused by transferring workload within the cluster can be reduced by limiting the number of cluster members, for example, through selecting cluster members with longer connection time to the cluster head.

In the following, we analyze the potential communication overhead in the clustering process. Assume that the basic safety message (BSM) as defined in SAE J2735 [48] is adopted to carry the broadcasting information of vehicles in each zone [48]. A broadcasting message contains the vehicle's available computation resource and the connection time to each of its neighbor vehicles. Assuming that a 4 Bytes field is padded for the information of 'available computation resource,' and another 4 Bytes field is padded for the information of 'connection time to a neighbor vehicle'. Then, the total length of a broadcasting message will be $(39 + 4 + 4 \times N)$ Bytes, where N is the number of neighbor vehicles, and the prefix of BSMs is 39 Bytes. Consider that the RSU has a coverage of 800 meters, which is further divided into 5 zones, then, each zone is 160 meters. With a higher vehicle density of 150 vehicles per kilometers, then, each zone has 30 neighbor vehicles. By setting $N = 30$, the total length of a broadcasting message is 163 Bytes. Considering a conservation data rate of 6 Mb/s as defined in the DSRC standards [50], the transmission time will be about 0.2 ms $(163 \times 8 \div 6)$. Therefore, the communication overhead in the clustering process is acceptable.

B. Evaluation of Mobile Resources

We consider the computation offloading problem in a specific scheduling duration $T = \{t_1, \ldots, t_{|T|}\}$, and the measurement unit is *second*. Denote O as the set of zones within the RSU's coverage, and $O = \{o_1, \ldots, o_Z\}$. As shown in Fig. 1, a cluster is formed within each zone o_z and acts as a powerful mobile edge server h_z. Let H represent the set of mobile edge servers. In the following, we describe how to evaluate the available computation resource of each mobile edge server.

Assume that each computation task has an expected deadline D, which follows the probability distribution function $P(d)$, and the measurement unit of the delay variable is *second*. Since

TABLE II
INFORMATION TABLE OF VEHICLE e_m

Metrics	1st neigh.	2nd neigh.	...	Nth neigh.
Conn. time	ct_1^m	ct_2^m	...	ct_N^m
Comp. freq.	cp_1^m	cp_2^m	...	cp_N^m
Cont. value	cv_1^m	cv_2^m	...	cv_N^m

each scheduling duration contains multiple discrete time slots, we assume that $P(d)$ is a discrete distribution function. Denote N as the number of one-hop neighbor vehicles of vehicle e_m. The connection time between e_m and its nth one-hop neighbor vehicle can be computed according to the two vehicles' driving speed, locations and directions. However, the *effective connection time* is the minimum value among the connection time and the shortest dwelling time of them within the RSU's coverage. In the following, if not particularly indicated, the connection time refers to the effective connection time, and is represented by ct_n^m. Given the connection time ct_n^m and the computation rate cp_n^m of the nth one-hop neighbor vehicle, we compute the 'contribution value' (cv_n^m) to evaluate the potential contribution of the nth one-hop neighbor vehicle on providing computation resource as a cluster member. Specifically, cv_n^m is defined as:

$$cv_n^m = \sum_{1 \le t \le ct_n^m} P(t) * t * cp_n^m, \tag{1}$$

where $t * cp_n^m$ is the available computation resource for completing a task with expected deadline t, $P(t)$ is the discrete probability value when the expected deadline is t. Note that the available service time of the nth one-hop neighbor vehicle is within the range of ct_n^m.

Each vehicle e_m maintains an information table of its one-hop neighbor vehicles as Table II. Then, the 'overall contribution value' (i.e., OCV_m) of vehicle e_m is defined as the summation of all the contribution values of its one-hop neighbor vehicles, given by

$$OCV_m = \sum_{1 \le n \le N} cv_n^m. \tag{2}$$

According to the clustering approach described in Section IV-A, after all vehicles in the same zone broadcasting their OCVs, the one with the highest OCV is selected as the cluster head, and its one-hop neighbor vehicles are identified as the cluster members. When receiving a computation task from the RSU, the cluster head will divide the task into multiple sub-tasks, and allocate them according to the connection time and the available computation resource of each cluster member. Assume that vehicle e_m is within the zone o_z, then, the formed cluster in zone o_z is also the mobile edge server h_z. The available computation frequency of h_z (i.e., V_z) is approximately computed as follows:

$$V_z = \sum_{1 \le n \le N} ct_n^m \cdot cp_n^m / dt_m, \tag{3}$$

where dt_m is the dwelling time of cluster head e_m within the RSU's coverage, which is also the available service time of mobile edge server h_z.

C. Formulation of LEGAP

Denote $A = \{a_1, \ldots, a_{|A|}\}$ as the set of computation tasks submitted in duration T by nearby users. Each task $a_j (1 \le j \le |A|)$ is characterized by a quadruples $\Omega_j = \{G_j, U_j, \tau_j, D_j\}$, which represents the size of input data, size of computation task, the arrival time and the expected deadline (i.e., the expected delay to complete the computation task), respectively. The measurement of G_j is the *bit*, and the measurement of U_j is the *cycles*. Let H, F and C represent the set of mobile edge servers, set of static edge servers and the cloud server, respectively. Then, all the computing servers can be represented by $S = H \cup F \cup C$. To facilitate description, let $S = \{s_1, \ldots, s_{|S|}\}$, where the indices of s_i from $1 \le i \le |H|$, $|H| + 1 \le i \le |H| + |F|$ and $i = |S|$ represent the mobile edge servers, the static edge servers and the cloud server, respectively. Consider that different computing servers have distinct features of communication/computation capabilities, the allocated transmission rates and computation frequencies from these computing servers are thus different for a specific computation task. Denote $v_{i,j}$ as the computation frequency (i.e., *the number of cycles per second*) of task a_j at computing server s_i. Let $r_{i,j}$ represent the allocated transmission rate (i.e., *bit per second*) for task a_j by the computing server s_i. Let $R = \{R_1, \ldots, R_{|S|}\}$ and $V = \{V_1, \ldots, V_{|S|}\}$ represent the maximum communication and computation capacities of computing servers, respectively. We introduce the variables $x_{i,j} (1 \le i \le |S|, 1 \le j \le |A|)$ to indicate whether task a_j is offloaded to computing server s_i. If a_j is offloaded to s_i, then $x_{i,j} = 1$; Otherwise, it is zero.

As suggested in many literatures [51], [52], the size of a computation result is very small compared with the size of the computation task, we ignore the transmission delay of computation results. If task a_j is offloaded to static edge server s_i ($|H| + 1 \le i \le |H| + |F|$), the service delay is calculated by:

$$d_{i,j}^f = \Delta_j + \frac{G_j}{r_{i,j}} + \frac{U_j}{v_{i,j}}, \tag{4}$$

where Δ_j is the waiting time of task a_j. $G_j / r_{i,j}$ is the transmission delay of task a_j from the user to edge server s_i. $U_j / v_{i,j}$ is the computation delay when task a_j is completed by edge server s_i. As described above, the static edge servers are near to the RSU and connected via the wired network, we assume that tasks can be directly offloaded to static edge servers via various communication interfaces (i.e., 5G/4G and WiFi, etc.).

If task a_j is offloaded to the mobile edge server s_i ($1 \le i \le |H|$), it will be first transmitted to the RSU, and then, the RSU transmits it to s_i. This is because that the users that need to offload tasks may not be in the communication range of the mobile vehicles. In addition, the communication connectivity between users and mobile vehicles are intermittent, which causes unstable computation offloading. Therefore, the users will offload tasks to mobile edge servers via the RSU, and the service delay is calculated by:

$$d_{i,j}^h = \Delta_j + \frac{G_j}{r_{0,j}} + \frac{G_j}{r_{i,j}^0} + \frac{U_j}{v_{i,j}} + d_i^{Trasf}, \tag{5}$$

where $r_{0,j}$ and $r^0_{i,j}$ are the transmission rates of task a_j from the user to the RSU, and from the RSU to the edge server s_i, respectively. d_i^{Trasf} is the transferring delay of the intra-cluster when coordinating cluster members to complete tasks in parallel. Considering the mobility of vehicles, the computation results will be returned back from s_i to the user via the RSU.

Otherwise, if task a_j is offloaded to the cloud server via the RSU, the transmission delay can be assumed as a constant when the RSU-cloud link is through the high-capacity wired optical network [53]. The computation delay at the cloud server can be ignored since it is generally equipped with powerful workload processing units [54], [56], and then, the total response time can be assumed as a constant σ [14], [45]. Therefore, the service delay at the cloud server is computed as:

$$d^c_{i,j} = \Delta_j + \frac{G_j}{r_{0,j}} + \sigma. \tag{6}$$

Denote $d_{i,j}$ as the service delay of task a_j when it is offloaded to any computing server s_i. Then,

$$d_{i,j} = \begin{cases} x_{i,j} \cdot d^f_{i,j}, |H|+1 \le i \le |H|+|F|, \\ x_{i,j} \cdot d^h_{i,j}, \quad 1 \le i \le |H|, \\ x_{i,j} \cdot d^c_{i,j}, \quad i = |S|. \end{cases} \tag{7}$$

We assume that the users pay for the computation offloading services if their tasks are completed within the required deadlines. From the perspective of the network infrastructure, without considering the computation cost, the RSU will receive the payments as its revenues and distribute them to computing servers involved in the computation offloading process. The service charging function $p_{i,j}$ for completing task a_j at computing server s_i can be defined with the service delay and the required resource amount. If the service delay is no longer than the expected deadline, the revenue is linearly increasing with the amount of required communication and computation resources. If the service delay is longer than the expected deadline but not longer than a tolerable latency, then, the task is processed with low quality of service (QoS). In this case, the revenue will be affected by the exceeded service delay. Otherwise, if the service delay is longer than the tolerable latency, the RSU will not receive any revenue. Therefore, we have the revenue function [56]:

$$p_{i,j} = \begin{cases} \alpha_1 \cdot C_j + \alpha_2 \cdot U_j, & \text{if } 0 < d_{i,j} \le D_j, \\ \alpha_1 \cdot C_j + \alpha_2 \cdot U_j - \beta \cdot (d_{i,j} - D_j), & \text{if } D_j < d_{i,j} \le \omega D_j, \\ 0, & \text{if } d_{i,j} = 0 \text{ or } d_{i,j} > \omega D_j, \end{cases} \tag{8}$$

where α_1 and α_2 are the unit price of the communication and computation resources, respectively. Parameter β indicates the penalty degree of exceeding the expected deadline. ω is the degree of tolerance. When the task is completed with low QoS, the revenue is decreasing linearly with service delay $d_{i,j}$.

To maximize the total revenue in entire scheduling duration, we formulate the following computation offloading problem.

$$\begin{aligned} \text{P1}: \ & \max_{\mathrm{x_{i,j}}} \sum_{\mathrm{i=1}}^{\mathrm{|S|}} \sum_{\mathrm{j=1}}^{\mathrm{|A|}} \mathrm{p_{i,j} \cdot x_{i,j}}, \\ \text{s.t.} \ & (\text{C1}): \sum_{j=1}^{|A|} r_{i,j} \cdot x_{i,j} \le R_i, i = |H|+1, \ldots, |H|+|F|, \\ & (\text{C2}): \sum_{j=1}^{|A|} v_{i,j} \cdot x_{i,j} \le V_i, i = 1, \ldots, |S|, \\ & (\text{C3}): dt_i \cdot x_{i,j} \ge d_{i,j}, i = 1, \ldots, |H|, j = 1, \ldots, |A|, \\ & (\text{C4}): \sum_{i=1}^{|S|} x_{i,j} \le 1, j = 1, \ldots, |A|, \\ & (\text{C5}): x_{i,j} \in \{0,1\}, i = 1, \ldots, |S|, j = 1, \ldots, |A|, \end{aligned} \tag{9}$$

where C1 is the constraint on communication resources of static edge servers. C2 is the constraint on computation resources of all the computing servers. We use C3 to guarantee that the available service time of any mobile edge server (i.e., the connection time between the mobile edge server and the RSU) should be larger than the service delay of offloaded tasks. C4 requires that each computation task can be allocated to at most one computing server. C5 is the constraint of binary variables. For unified description, C1 and C3 are slightly changed as follows. First, since C1 only constrains the tasks that are offloaded to the static edge server, we set $r^0_{i,j} = 0$ if task a_j is offloaded to the mobile edge server (i.e., indexed by $1 \le i \le |H|$) and the cloud server (i.e., indexed by $i = |S|$). Second, for any pair of task a_j and computing server s_i, if C3 is not satisfied, we set $v_{i,j} = \infty$ to guarantee that a_j will never be allocated to mobile edge server s_i.

The above problem is a 'less than or equal to' generalized assignment problem (LEGAP), where 'less than or equal to' means that a task can be offloaded to at most one computing server. It differentiates the generalized assignment problem (GAP) which requires that one item should be put into exactly one knapsack (i.e., the sum of binary variables in C4 should be equal to '1') [57]. Actually, LEGAP is a special class of the multi-dimensional multiple knapsack problem (MMKP), where the computing servers are considered as knapsacks with two-dimensional features in terms of communication and computation capabilities. The computation tasks are treated as items which will consume different amount of resources when they are offloaded to different computing servers. However, in the MMKP, the revenue received by offloading a task is independent of the computing server, while in LEGAP, it receives different revenues when the task is offloaded to different computing servers.

V. Proposed Algorithms

In the formulated problem (9), by setting $\Delta_j = T - \tau_j$, it becomes a periodical scheduling problem where a set of tasks wait at the specific time to be scheduled, and the required information of all the tasks are known to the RSU. For the periodical scheduling problem, we propose an offline bound-and-bound optimal (BBO) algorithm to maximize the system revenues in Section V-A. Otherwise, by setting $\Delta_j = 0$, it becomes an online

Algorithm 1: The LOWER Algorithm.

Input: Task set A, computing server set K, revenue set P, transmission rate $r_{i,j}$ and computation capability $v_{i,j}$ from a_j to s_i; transmission rate R_i and computation capacity V_i; stack S_i and current solution $[\hat{x}]$; current computing server m

Output: Lower bound L and corresponding solution $[\tilde{x}]$

Steps:

1 $L = \sum_{i=1}^{m} \sum_{j \in Q_m} p_{i,j} \cdot \hat{x}_{i,j}$;
2 $N' = \{j | \hat{x}_{i,j} = 0, i = 1, ..., m\}$;
3 $\bar{N} = N' \backslash Q_m$;
4 $\bar{R} = R_i - \sum_{j \in Q_m} r_{m,j} \cdot \hat{x}_{m,j}$;
5 $\bar{V} = V_i - \sum_{j \in Q_m} v_{m,j} \cdot \hat{x}_{m,j}$;
6 $i = m$;
7 **while** $i \leq |S|$: **do**
8 Solve the two-dimensional single knapsack problem by allocating items in $\bar{N}$, with the resource constraints $\bar{R}$ and $\bar{V}$. Store the solution vector in row i of $[\tilde{x}]$.
9 $L = L + \bar{z}$.
10 $N' = N' \backslash \{j | \tilde{x}_{i,j} = 1\}$.
11 $\bar{N} = N'$.
12 $i = i + 1$.
13 $\bar{R} = R_i$, $\bar{V} = V_i$.
14 **end while**

Algorithm 2: The UPPER Algorithm.

Input: Task set A, computing server set S, revenue set P, transmission rate $r_{i,j}$ and computation capability $v_{i,j}$ from a_j to s_i; transmission rate R_i and computation capacity V_i; stack Q_i and current solution $[\hat{x}]$; current computing server m

Output: Upper bound U_0

Steps:

1 $\bar{R} = R_m - \sum_{j \in Q_m} r_{m,j} \cdot \hat{x}_{m,j} + \sum_{i=m+1}^{|S|} R_i$;
2 $\bar{V} = V_m - \sum_{j \in Q_m} v_{m,j} \cdot \hat{x}_{m,j} + \sum_{i=m+1}^{|S|} V_i$;
3 Remove the classes from which a task has been chosen. That is, $\bar{N} = \{N_j | x'_{i,j}{=}0, \forall i \in N_j\}$;
4 Solve the MCKP by allocating items in $\bar{N}$, with the resource constraints $\bar{R}$ and $\bar{V}$.
5 $U = \sum_{i=1}^{m} \sum_{j \in Q_i} p_{i,j} \cdot \hat{x}_{i,j} + \bar{z}$.

scheduling problem where a task is scheduled without waiting other tasks, and the required information of the following tasks are unknown to the RSU. For this problem, we adopt an online heuristic algorithm to make real-time offloading decisions in Section V-B. The online heuristic algorithm can guarantee that the resource capacities of all the computing servers are never exceeded with new tasks arriving.

A. The BBO Algorithm

The main idea of the traditional branch-and-bound algorithm and its extensions is that, each time when inserting an item into the current knapsack ('1' branch) or not ('0' branch), it derives the upper and lower bounds under this branch. Other branches that generate higher upper bounds will not be considered because the resulted solution space cannot contain the optimal solution. The branching and backtracking operations continue until the lower bound is improved up to the upper bound. We first compute the upper bound and the low bound in Section V-A1 and Section V-A2, respectively. In Section V-A3, we describe the detailed procedure of the modified bound-and-bound algorithm.

1) Upper Bound: To retrieve a tight initial upper bound of problem P1, we first use the surrogate relaxation [60] to compute an initial upper bound U_1, and then, we use the UPPER algorithm in Table V (as seen in Appendix A) to compute another initial upper bound U_0. Finally, the tight initial upper bound of P1 (i.e., UB) can be obtained by choosing the smaller one of U_1 and U_0.

First, we compute the initial upper bound U_1 as follows. Let $\mu = (\mu_1, \ldots, \mu_{|S|})$ and $\sigma = (\sigma_1, \ldots, \sigma_{|S|})$ be two vectors of nonnegative multipliers corresponding to the communication and computation resource constraints. Without considering constraint C3 in P1, the relaxed problem $SP(\mathrm{P1}, \mu, \sigma)$ is then given by:

$$\begin{aligned} \mathrm{P2}: \ & \mathrm{z}\left(\mathrm{SP}\left(\mathrm{P1}, \mu, \sigma\right)\right) = \max_{\mathrm{x_{i,j}}} \sum_{\mathrm{i=1}}^{\mathrm{|S|}} \sum_{\mathrm{j=1}}^{\mathrm{|A|}} \mathrm{p_{i,j}} \cdot \mathrm{x_{i,j}}, \\ \text{s.t.} \ & \sum_{i=1}^{|S|} \mu_i \sum_{j=1}^{|A|} r_{i,j} \cdot x_{i,j} \leq \sum_{i=1}^{|S|} \mu_i R_i, \\ & \sum_{i=1}^{|S|} \sigma_i \sum_{j=1}^{|A|} v_{i,j} \cdot x_{i,j} \leq \sum_{i=1}^{|S|} \sigma_i V_i, \\ & \sum_{i=1}^{|S|} x_{i,j} \leq 1, j = 1, \ldots, |A|, \\ & x_{i,j} \in \{0,1\}, i = 1, \ldots, |S|, j = 1, \ldots, |A|. \end{aligned} \tag{10}$$

According to [59], since for any instance of the multiple knapsack problem (MKP), the optimal choice of multipliers $\mu_1, \ldots, \mu_{|S|}$ in the surrogate relaxation problem can be set to any positive constant. For simplicity, we set the multiplier vectors μ and σ as $\mu_i = c_1$ and $\sigma_i = c_2$ for $i = 1, \ldots, |S|$, where c_1 and c_2 are positive constants. Let the binary variable $x'_j = \sum_{i=1}^{|S|} x_{i,j}$ indicate whether task a_j is offloaded to any computing server $s_i (i = 1, \ldots, |S|)$. By considering the communication and computation resource constraints, respectively, we get two relaxed subproblems as follows:

$$\begin{aligned} \mathrm{P3-1}: \ & \max_{\mathrm{x'_j}} \sum_{\mathrm{j=1}}^{\mathrm{|A|}} \mathrm{p'_j} \cdot \mathrm{x'_j}, \\ \text{s.t.} \ & \sum_{j=1}^{|A|} r'_j \cdot x'_j \leq R, \\ & x'_j \in \{0,1\}, j = 1, \ldots, |A|, \end{aligned} \tag{11}$$

$$\begin{aligned} \mathrm{P3-2}: \ & \max_{\mathrm{x'_j}} \sum_{\mathrm{j=1}}^{\mathrm{|A|}} \mathrm{p'_j} \cdot \mathrm{x'_j}, \\ \text{s.t.} \ & \sum_{j=1}^{|A|} v'_j \cdot x'_j \leq V, \\ & x'_j \in \{0,1\}, j = 1, \ldots, |A|, \end{aligned} \tag{12}$$

where $p'_j = \max\{p_{i,j}|i = 1, \ldots, |S|\}$, $r'_j = \min\{r_{i,j}|i = 1, \ldots, |S|\}$, $R = \sum_{i=1}^{|S|} R_i$, $v'_j = \min\{v_{i,j}|i = 1, \ldots, |S|\}$ and $V = \sum_{i=1}^{|S|} V_i$. Let zr^* and zv^* represent the optimal solutions of (11) and (12). Then, we compute the upper bound as $U_1 = \min\{zr^*, zv^*\}$.

Second, we use the UPPER algorithm to compute another initial upper bound U_0. The basic idea of the UPPER algorithm is to simplify the MKP as a single knapsack problem by accelerating the available resources of all the computing servers together into one powerful computing server, and assigning the remaining tasks optimally into the powerful computing server. When assigning the task to a large computing server, since it is not clear which specific computing server the task is allocated to, the amount of consumed resources and the received revenue are unknown. Therefore, we interpret the above problem as follows. Consider $|A|$ mutually disjoint classes $N_1, \ldots, N_{|A|}$ of tasks, each class N_j has $|S|$ tasks. For any task $a'_{i,j} \in N_j$, the required communication resource $r'_{i,j}$, computation resource $v'_{i,j}$, and its revenue $p'_{i,j}$ satisfy that $r'_{i,j} = r_{i,j}$, $v'_{i,j} = v_{i,j}$ and $p'_{i,j} = p_{i,j}$. The problem is transformed into selecting at most one task from each class for a large computing server, such that the sum of revenues is maximized without exceeding the resource capacity constraints. The communication and computation resource capacities of the large computing server are $\bar{R} = \sum_{i=1}^{|S|} R_i$ and $\bar{V} = \sum_{i=1}^{|S|} V_i$, respectively. We introduce the binary variables $x'_{i,j}$, which equals '1' if and only if task $a'_{i,j}$ is selected. Then, the problem is formulated as:

$$\begin{aligned}
\mathrm{P4}: \quad & \max_{\mathrm{x'_{i,j}}} \sum_{\mathrm{j=1}}^{|\mathrm{A}|} \sum_{\mathrm{i \in N_j}} \mathrm{p'_{i,j}} \cdot \mathrm{x'_{i,j}}, \\
\text{s.t.} \quad & (\mathrm{C6}): \sum_{j=1}^{|A|} \sum_{i \in N_j} r'_{i,j} \cdot x'_{i,j} \leq \bar{R}, \\
& (\mathrm{C7}): \sum_{j=1}^{|A|} \sum_{i \in N_j} v'_{i,j} \cdot x'_{i,j} \leq \bar{V}, \\
& (\mathrm{C8}): \sum_{i \in N_j} x'_{i,j} \leq 1, j = 1, \ldots, |A|, \\
& (\mathrm{C9}): x'_{i,j} \in \{0, 1\}, j = 1, \ldots, |A|, i \in N_j.
\end{aligned} \tag{13}$$

If we replace constraint C8 by $\sum_{i \in N_j} x'_{i,j} = 1$, P4 will become a multiple choice knapsack problem (MCKP). We can achieve this by adding a dummy task $a'_{i+1,j}$ with $(r'_{i+1,j}, v'_{i+1,j}, p'_{i+1,j}) := (0, 0, 0)$ to each class N_j. Then, we use the branch-and-bound algorithm in [59] to solve the MCKP. The obtained objective value is recorded as the upper bound U_0.

Finally, we take the smaller one of U_1 and U_0 as the tight initial upper bound of LEGAP, i.e., $UB = \min\{U_1, U_0\}$. Note that in the following branching process, the UPPER algorithm will also be called to compute the upper bound at each branch node.

2) Lower Bound: In addition to the upper bound, we compute a lower bound at each branch node via the heuristic LOWER algorithm (as shown in Appendix B). Then, the next branching is conducted according to the current heuristic solution. According to [57], the heuristic solution is expected to better guide the branching process than random or sequential choices for each branch node.

The basic idea of the LOWER algorithm is to sequentially solve $|S|$ individual knapsack problems as follows. We first compute the approximate resource capacity of any computing servers s_i as $C_i = R_i \cdot V_i$, and sort the computing servers in increasing order of the approximate capacity. The first computing server $s_i = 1$ is loaded optimally by solving a two-dimensional knapsack problem. Then, the second computing server $s_i = 2$ is loaded optimally by selecting from the remaining tasks. Only when the current computing server is completely loaded, the next computing server will be loaded with the remaining tasks. Besides, in each iteration, the tasks that have been selected are no longer considered for the following computing servers. This process continues until all the $|S|$ computing servers have been loaded.

3) Bound-and-Bound Process: In the bound-and-bound procedure, we use the LOWER algorithm to compute a new lower bound at the current branch node, and the following branching process is conducted by the heuristic solution. The UPPER algorithm computes an upper bound at each branch node and checks if a better solution is generated. The branching process is described as follows:

- First, we compute the initial upper bound of LEGAP using the surrogate relaxation method and the UPPER algorithm. The LOWER algorithm is applied to compute the initial heuristic solution and the corresponding lower bound.
- Second, the branching process is triggered based on the heuristic solution. For each allocation variable which equals to '1,' we generate two branching nodes associated with the allocation variable. One is to follow the original solution (i.e, '1'), that is, allocating the task to the corresponding computing server. The other is to exclude the task from the corresponding computing server. Then, the remaining available resources of the computing server will change according to different branching operations.
- Third, the UPPER algorithm is called again under the current branching operation to compute the new upper bound. If the upper bound is lower than the current optimal solution, a backtrack process will be conducted.
- Fourth, for each '0' branch, the LOWER algorithm is called to obtain the new greedy solution and lower bound. The initial lower bound is updated when a larger bound is generated. We also update the current optimal solution as the new heuristic solution.

Based on above descriptions, since the LOWER algorithm does not prune the search space of the optimal solutions, and the UPPER algorithm gradually selects better solutions, it can be proved that the bound-and-bound procedure is able to derive the exact solution of the LEGAP [59]. Due to limited space, the detailed algorithm descriptions are given in Appendix C.

Besides, since the BBO algorithm is a variant of the traditional branch-and-bound algorithm, and the worst-case time complexity is exponential. However, the practical execution time cannot be per-determined because the number of branching depends on the specific instance. The BBO algorithm can be used for comparison in small instances of computation offloading. For the large-scale instances, we can preset an execution time to avoid excessively long execution time.

4) A Toy Example: We give a simple example to illustrate the bound-and-bound procedure. For simplicity, we set a non-zero array to the transmission rates from users to different computing servers, and set a non-zero array to the allocated computation frequencies for all tasks at different computing servers. Given 6 tasks and 2 computing servers, the transmission rates of different tasks to the computing servers are $[4, 2, 5, 4, 4, 9; 1, 5, 6, 7, 5, 8]$ and the measurement is *Mbps*. The allocated computation frequencies of different tasks at the computing servers are $[3, 6, 4, 5, 1, 3; 8, 4, 4, 6, 4, 5]$, and the measurement is *Gcycles/s*. Assume that the revenues between each pair of tasks and computing servers have been computed according to the resource requirements and expected deadlines of the tasks, and are $[6, 7, 2, 8, 10, 3; 3, 4, 7, 2, 5, 9]$. The maximum transmission rates between users and the two computing servers are 12 Mbps and 9 Mbps, respectively. The maximum computation frequencies of the two computing servers are 14 Gcycles/s and 15 Giga cycles/s, respectively.

We describe the branching process as follows. First of all, at the root node, the upper and lower bound are computed. We choose the branch variable $\hat{x}_{1,2} = 1$ from the heuristic solution. The branching process continues until the three tasks (i.e., a_2, a_4 and a_5) in the first knapsack are completely checked. Then, we can find that the optimal value z and upper bound U_0 are equal. This is because the LOWER algorithm fills the knapsack one by one through optimally selecting tasks from the remaining tasks. The UPPER algorithm computes the upper bound by accelerating the resources into a large knapsack. At branch node 3, only the second knapsack is filled with the remaining tasks. Therefore, the two algorithms derive the same solution. Next, the algorithm goes backtrack and pops task a_5. New heuristic solution is generated, as well as new branch variable $\hat{x}_{1,1}$. However, the updated upper bound U_0 at branch $\hat{x}_{1,1} = 1$ is lower than the current optimal value z. Therefore, the second backtrack process is called. The iteration continues until $z = UB$. In this example, $z = UB$ can not be obtained since the solution associated with UB is not feasible. Therefore, the branch tree will be completely traversed and the optimal value is exactly the lower bound L.

B. Online Heuristic Algorithm

In the online scheduling, the computation offloading decisions are made at the arrival of tasks. Without knowing the required resources and the expected deadlines of future tasks, the scheduling algorithm decides to reject the offloading requests or where to offload the computation task according to the remaining resources of computing servers, and the required resources of the current task.

The online heuristic algorithm adopts a threshold-based scheduling approach. When task a_j is offloaded to computing server s_i, we define the communication efficiency $e^1_{i,j}$ as the ratio of revenue to the required communication rate. Similarly, the computation efficiency $e^2_{i,j}$ is defined by the ratio of revenue to the demanding computation frequency:

$$e^1_{i,j} = \frac{p_{i,j}}{r_{i,j}}, \tag{14}$$

$$e^2_{i,j} = \frac{p_{i,j}}{v_{i,j}}. \tag{15}$$

Whenever task a_j arrives, let r_i and v_i represent the fraction of currently occupied communication and computation resources of node s_i, respectively. Denote E_j as the set of available computing servers to which task a_j can be allocated, and then we have:

$$E_j = \left\{ s_i \left| e^1_{i,j} \geq \Psi\left(r_i\right), e^2_{i,j} \geq \Psi\left(v_i\right) \right. \right\}, \tag{16}$$

where r_i and v_i are the fractions of occupied resources ($0 \leq r_i, v_i \leq 1$). $\Psi(r_i)$ and $\Psi(v_i)$ are functions in terms of r_i and v_i. We allocate task a_j to the computing servers in set E_j with the maximum $p_{i,j}$, as long as it satisfies the constraints of communication and computation capacities.

As given in [60], for the fraction r_i of the communication resource occupied already, $\Psi(r_i)$ can be set as:

$$\Psi\left(r_i\right) = \left(U^r e / L^r\right)^{r_i} \left(L^r / e\right), \tag{17}$$

where e denotes the base of the natural logarithm. The positive constants L^r and U^r are the lower and upper bounds of the communication efficiencies of all tasks, respectively, i.e.,

$$L^r \leq e^1_{i,j} \leq U^r, \forall i = 1, \ldots, |S|, j = 1, \ldots, |A|. \tag{18}$$

We set $\Psi(v_i)$ in the same way. Then, given a constant c^r where

$$c^r = \frac{1}{1 + \ln\left(U^r / L^r\right)}, \tag{19}$$

we can observe that for variable $r_i \in [0, c^r]$, it satisfies that $\Psi(r_i) \leq L^r$. Therefore, the algorithm will pick all tasks which meet the resource constraints until c^r fraction of the communication resource or c^v fraction of the computation resource are occupied. In addition, when $r_i = 1$, we have $\Psi(i_i) = U^r$; when $v_i = 1$, we have $\Psi(v_i) = U^v$. Since $\Psi(r_i)$ and $\Psi(v_i)$ are strictly increasing, the available resource capacities of computing servers will be never over-loaded. Therefore, with new tasks arriving, the online heuristic algorithm is able to schedule tasks based on the updated computation offloading environment in the current scheduling period. Based on [60], for one-dimensional online GAP, if the ratio of the jth item's weight $w_{i,j}$ to the capacity of the ith knapsack problem B_i is quite small, i.e., $w_{ij}/B_i \leq \varepsilon (\varepsilon > 0)$, and ε is very close to 0, then, the competitive ratio of the threshold-based algorithm can achieve up to $(\ln(U/L) + 2)$.

The time complexity of the proposed online heuristic algorithm is $O(n^2)$. Two 'for' loops traverse all the real-time arrived computation tasks and the computing servers to decide the offloading solutions according to the required resources and the expected deadlines of the tasks, as well as the currently occupied resources of the computing servers.

VI. Simulation Results and Analysis

To compare the performances of the BBO and the online heuristic algorithms, we implement the other four online algorithms, including the DRL-based algorithm, the random algorithm, the Revenue_First algorithm and the R2C_First algorithm. The online DRL-based scheduling model takes the

TABLE III
SIMULATION PARAMETERS

Parameter	Value
Each scheduling duration	50 ms
Arrival rate of tasks	16 tasks/10 ms
Number of zones in a RSU	5 [48]
Number of static edge nodes	4
Size of input data	$[300, 500]$ Bytes [14]
Size of computation task	$[200, 300]$ Kcycles [14]
α_1	5 cents/10^6 cycles [58]
α_2	0.5 cents/Mb
β, ω	0.2, 1.2 [58]
Expected deadline of each task	$[60, 80]$ ms
Coverage of a RSU	800 m [63]
Density of vehicles	60 vehicles/km
Demanding transmission rates of each task	$[1, 10]$ Mbps
Demanding computation rate for each task	$[1, 2]$ Gcycles/s [64]
Average computation rate of each vehicle	0.8 Gcycles/s [5]
Maximum computation rate of static edge servers	$[5, 8]$ Gcycles/s [5]

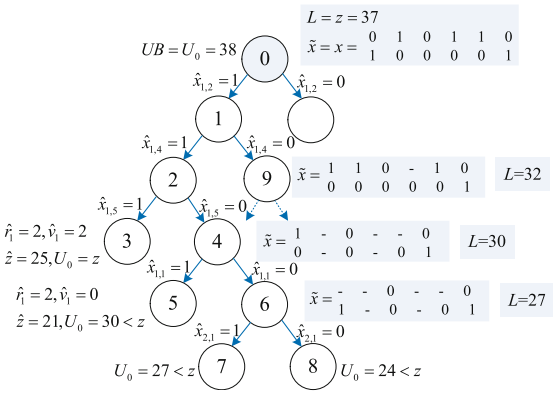

Fig. 2. An example of bound-and-bound procedure.

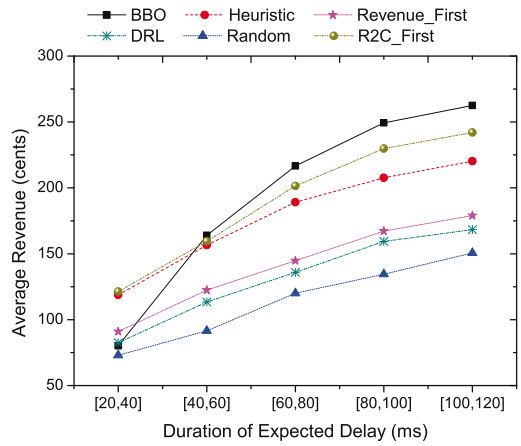

Fig. 3. Average revenues under different expected deadlines.

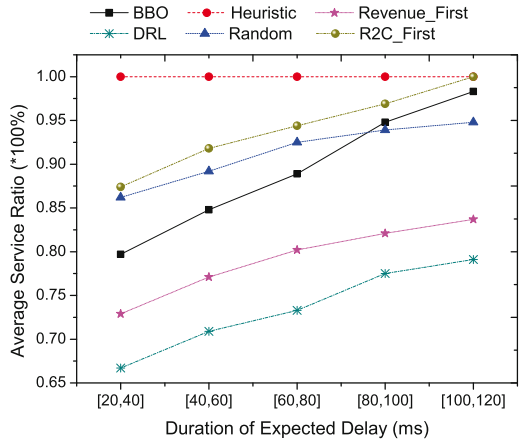

Fig. 4. Average service ratios under different expected deadlines.

current resource capacities of all computing servers and the requirement information of the task as the input, and outputs the offloading decision for this task. The online random algorithm makes offloading decision by randomly choosing a computing server with adequate resource capacities for the task. The online Revenue_First algorithm schedules the task to the available computing server from which the highest revenue is received. In addition, the online R2C_First algorithm schedules the task to the available computing server on which the task has the highest revenue-to-cost (R2C) ratio, which is calculated as the ratio of the received revenue to the communication and computation resource consumption. All the algorithms are implemented with Python. The simulation parameters are summarized in Table III. We compare the six algorithms with respect to the average revenue (i.e., computed by (8)) and the average service ratio. The former indirectly reflects the service delay, while the latter counts the number of tasks served within a tolerable delay (i.e., ωD_j).

A. *Effect of Expected Deadlines*

Fig. 3 compares the average revenues of the six algorithms under different settings of tasks' expected deadlines. With the increasing of expected deadlines, the average revenues of all the algorithms are gradually increasing due to the increased chances of completing the tasks. When the expected deadlines of the tasks are lower than 40 ms, the average revenues of the BBO are lower than that of the online algorithms (i.e., the heuristic one, DRL-based, Revenue_First, R2C_First). This is probably because that the BBO makes periodical computation offloading decisions at the default scheduling interval of 50 ms. It requires that the tolerant delay of the task is not exceeded at the specific scheduling time slot. Therefore, the task that can be processed timely should have both the longer expected deadline and the suitable arrival time at the RSU. In this case, the chances of effective computation offloading are reduced, which decreases the offloading revenue of the BBO. However, with the advantage of the centralized optimal scheduling, the BBO achieves higher revenues than the other online algorithms in most cases, and the online R2C_First algorithm ranks next to it. In addition, the online DRL-based algorithm obtains the slightly lower revenues compared with the online Revenue_First algorithm, and the online random algorithm have lowest revenues in all cases.

Fig. 4 compares the average service ratios of the six algorithms under different settings of expected deadlines. As shown, the service ratios of all the algorithms are increasing with the expected deadlines, and the online heuristic algorithm has the best performances than the other algorithm in all cases. This is mainly because that the online heuristic algorithm schedules

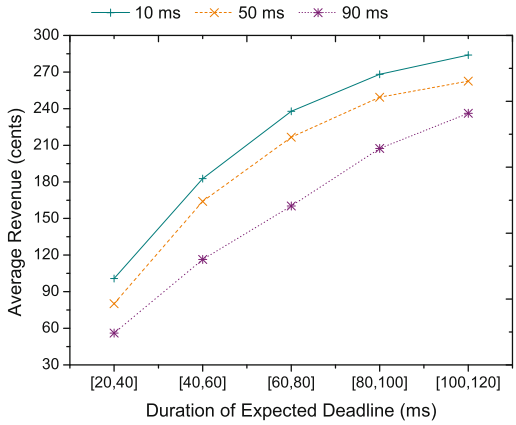

Fig. 5. Average revenues under different lengths of scheduling period.

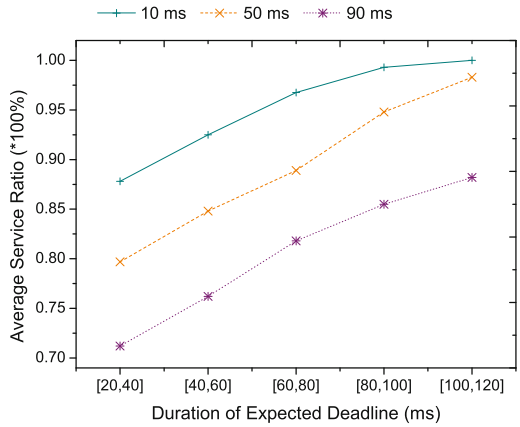

Fig. 6. Average service ratios of different lengths of scheduling period.

tasks by considering the fraction of the occupied resource of each computing server. However, refer to (8), the tasks that are completed within the expected deadlines (i.e., D_j) have higher revenues compared to the tasks that are completed within $[D_j, wD_j]$. Combining the results of Fig. 3 and Fig. 4, we can observe that the BBO schedules to complete more tasks without exceeding the expected deadlines, while the online heuristic algorithm schedules more tasks instantly with longer tolerant delays. In addition, although the online R2C_First and the online random algorithm have higher service ratios than the BBO, the latter has higher increasing rate than the former. Besides, the online DRL-based scheduling algorithm can achieve a relatively higher revenue than the online random algorithm at the sacrifice of the service ratio.

B. Effect of Length of Scheduling Period

Fig. 5 shows the average revenues of the BBO under different settings of scheduling periods. The default scheduling period is set to 50 milliseconds (i.e., shown in orange dashed line). In this test, we set the scheduling period to 10 ms (i.e., dark cyan solid line) and 90 ms (i.e., purple dotted line), respectively. As shown, the average revenues under shorter scheduling period (i.e., 10 ms) are higher than that under longer scheduling periods (i.e., 50 ms and 90 ms) in all settings of expected deadlines. However, within shorter scheduling period, the computation complexity is increased due to frequently scheduling of the centralized optimal algorithm.

Fig. 6 shows the average service ratios of the BBO under different settings of scheduling periods. Similar to Fig. 5, the service ratio under a shorter scheduling period (i.e., 10 ms) is apparently higher than that under longer scheduling periods (i.e., 50 ms and 90 ms). In addition, with the increasing of expected deadlines, both the average service ratio and the average revenue are improved due to the increasing ratio of successfully offloaded tasks. When the expected deadline is within the range of $[60, 80]$ ms, the service ratio under 10-ms scheduling period reaches up to 95%.

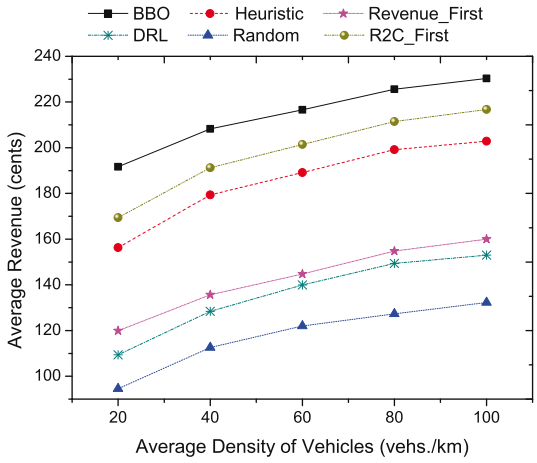

Fig. 7. Average revenues under different vehicle densities.

C. Effect of Vehicle Density

Fig. 7 compares the average revenues of the six algorithms under different vehicle densities. With the increasing of vehicle densities, the available computation resources of mobile edge servers are increased, and hence, the revenues achieved by different algorithms are gradually increasing. In accordance to Fig. 3, the BBO achieves the highest revenues compared with the other online algorithms, and the online R2C_First algorithm ranks next to it. The online DRL-based algorithm achieves the approximate performance with the online Revenue_First algorithm. This is probably because that the DRL model is prone to choose the computing server from which the maximum revenue is received by offloading the task. Different from other online algorithms which select the computing server according to the received revenue (i.e., online Revenue_First algorithm), the ratio of revenue to resource cost (i.e., online R2C_First algorithm), the fraction of occupied resources of computing servers (i.e., online heuristic algorithm), the online random algorithm selects the computing server randomly for each task, and hence, has the lowest revenues in all settings.

Fig. 8 compares the average service ratios of the six algorithms under different vehicle densities. Similar to Fig. 4, the online heuristic algorithm achieves the highest average service ratios

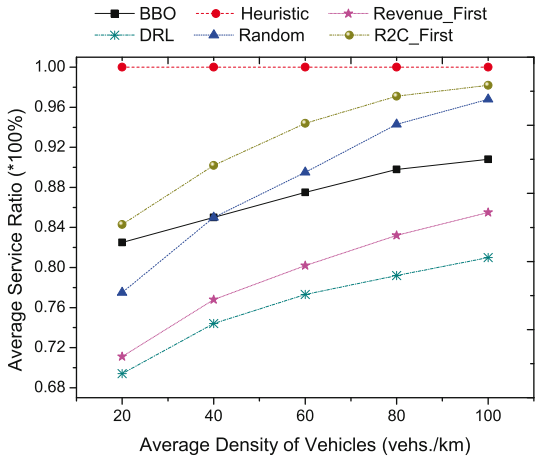

Fig. 8. Average service ratios under different vehicle densities.

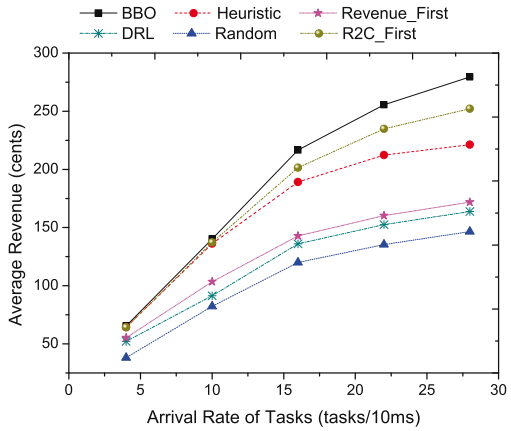

Fig. 9. Average revenues under different arrival rates.

in all settings. This is mainly because that the online heuristic algorithm selects the computing server by considering the fraction of the occupied resources, which facilitates the full utilization of communication/computation resources. Combining the results of Figs. 3, 4, 7 and 8, the centralized optimal scheduling algorithm (i.e., BBO) can achieve the highest revenues and relatively lower service ratios, and the online heuristic algorithm achieves the highest service ratios and relatively lower revenues. From the revenue function (8), it can be found that the BBO is prone to schedule the tasks within their expected deadlines, and the online heuristic algorithm is able to instantly schedule more tasks with longer delays. In addition, it can be observed that the online R2C_First algorithm can achieve both relatively high revenues and service ratios. Moreover, with the increasing of vehicle density, the service ratio of the BBO increases at slower speed than the other four online algorithms. Therefore, it can be inferred that the online algorithms can make better utilization of the increased computation resources by instantly offloading tasks to the available computing servers.

D. Effect of Task Arrival Rates

Fig. 9 compares the average revenues of the six algorithms under different arrival rates of tasks. With the increasing arrival

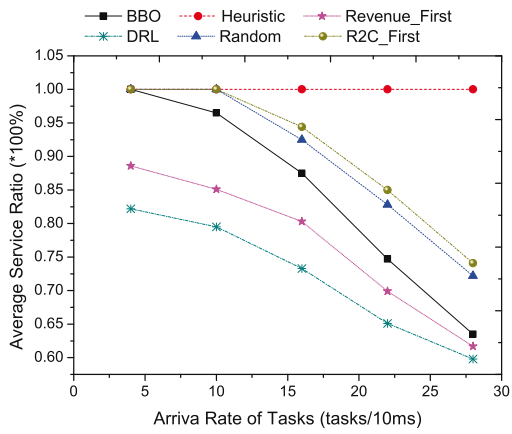

Fig. 10. Average service ratios under different arrival rates.

rates, all the algorithms are able to schedule more tasks and achieve higher revenues. The increasing rates of all the online algorithms are becoming slower under the default settings. This is probably because that, even the online algorithms are able to schedule tasks at their arrivals, the tasks are completed with longer delays due to limited resource capacities, and hence, the average revenue increases at a slower speed. Similar to Figs. 3 and 7, the BBO outperforms the other online algorithms and the online R2C_First algorithm ranks next to it. In addition, the online DRL-based algorithm and the online Revenue_First algorithm achieve the approximate revenues in all settings.

Fig. 10 compares the average service ratios of the six algorithms under different arrival rates of tasks. Similar to Figs. 4 and 8, the online heuristic algorithm achieves the highest service ratios in all scenarios, while the other algorithms have decreasing service ratios with the increased arrival rates. This is mainly because that the total resource capacities in the default settings are not adequate for severing the growing workload. Besides, the achieved service ratio by the BBO decreases more quickly than the other online algorithms. The main reason may be that the centralized optimal algorithm can not deal with the continuously increasing workload due to periodical scheduling scheme, which causes long waiting time for numerous tasks. Furthermore, combining with Fig. 9, although the service ratio of the BBO is decreasing quickly, it is still able to achieve the highest revenue by scheduling tasks with shorter delays.

E. Effect of Number of Zones

Fig. 11 and Fig. 12 show the average revenues and the service ratios of the six algorithms under different number of zones divided within the RSU's coverage. With the increasing number of zones, the available mobile edge servers are increased but the computation resource of each mobile edge server is reduced. As shown, the average revenues and the service ratios of all the algorithms are improved with the increasing number of zones. This is because that the mobile computation resources are utilized more efficiently by increasing the number of zones. However, the increasing rates of both the average revenue and the service ratio are becoming slower and tend to be stable due to the limited total computation resources within the RSU's coverage. In addition,

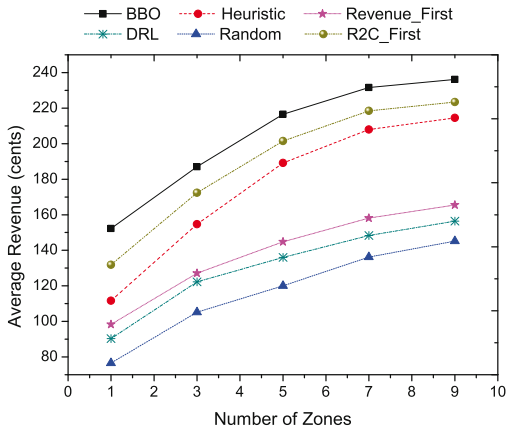

Fig. 11. Average revenues under different number of zones.

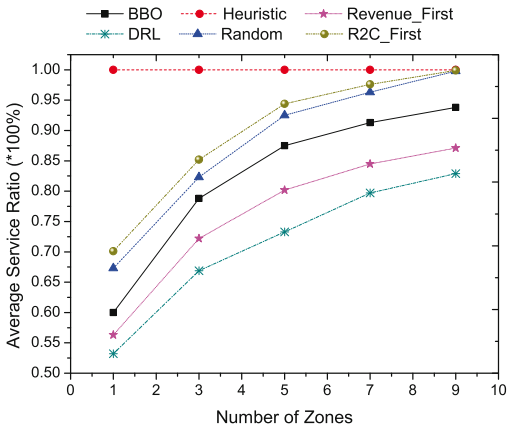

Fig. 12. Average service ratios under different number of zones.

as shown in Fig. 11, the average revenues of the BBO, the online R2C_First and the online heuristic algorithms are increasing more quickly than that of the online Revenue_First, online random and online DRL-based algorithms. Therefore, given the total resource capacity, the resource division and utilization have the varying degree influences on the performances of different algorithms. The optimal zone division can be achieved through experimental simulations under different parameter settings, so as to reach the best utilization of the mobile computation resources.

VII. Conclusions and Future Work

In this paper, we have presented a clustering-based cooperative computation offloading framework including the mobile/static edge servers and the cloud server. A distributed dynamic clustering approach has been proposed to classify moving vehicles into multiple cooperative edge servers. We have formulated the LEGAP to maximize the total revenues in real-time offloading environment. Further, we have proposed the offline BBO to schedule tasks periodically, where the upper bound at each branch node is computed by transforming the allocation subproblem into a MCKP. In addition, we have presented an online heuristic algorithm to instantly allocate or discard tasks at their arrivals with the resource capacity guarantees. Compared with online DRL-based algorithm, online random algorithm, online Revenue_First algorithm and online R2C_First algorithm, the proposed BBO can achieve the highest revenues and the proposed online heuristic algorithm greatly improves the service ratios.

In the future work, we will use other learning-based algorithms to predict the resource requirement statistics of the real-time offloading requests, and further improve the scheduling performances of online learning-based algorithms. Meanwhile, it would be a meaningful extension by utilizing the federated learning to realize the distributed cooperative computing in large-scale offloading environments, where each edge server can update the learning models independently without uploading the local computation tasks to a centralized resource scheduler.

Appendix A

Table V shows the details of UPPER algorithm. The stack $Q_i (i = 1, \ldots, |S|)$ contains the items that are assigned to knapsack s_i or excluded from it at the current branch. $[\hat{x}]$ represents the current solutions. m is the index of the current computing server. When computing the initial upper bound, the input of stack Q_i for any $i = 1, \ldots, |S|$ is empty, and the current computing server is $m = 1$ (i.e., s_1). In the bound-and-bound procedure, if the first $m - 1$ computing servers have been allocated with tasks, the remaining computing servers from m to $|S|$ are considered as a large computing server. The UPPER algorithm will assign the remaining tasks to this large computing server by solving a MCKP. Note that when UPPER algorithm is called at different branch nodes, the inputs of UPPER algorithm (i.e., the current stack Q_i, solution $[\hat{x}]$ and computing server m) are varying since different tasks have been allocated to different computing servers at the current branch. The upper bound associated with the specific branch node is thus generated.

Appendix B

Table IV shows the proposed algorithm. Line 1 computes the total revenues of tasks in stack Q_i where $i = 1, \ldots, m$. From line 2 to 5, the remaining tasks and the remaining resource amount of the mth computing server are calculated. From line 7 to 14, the computing server from m to $|S|$ are optimally loaded one by one. For each computing server, we solve a two-dimensional single knapsack problem via a simple branch-and-bound algorithm. Once a computing server is loaded, the obtained revenue is added to L, and the selected tasks are removed from the remaining tasks.

Appendix C

Table VI shows the details of the bound-and-bound procedure. Step 1 initializes the stack Q_i, the allocation variables $\hat{x}_{i,j}$ and the optimal value z that have been determined at the current branch. The upper bound UB is computed via comparing the UPPER algorithm and the surrogate relaxation. Step 2 describes a heuristic procedure. It calls the LOWER algorithm to compute lower bound L and the heuristic solution $[\tilde{x}]$. The current optimal solution $[x]$ is updated according to the allocation variables

TABLE IV
THE LOWER ALGORITHM

Algorithm 1 The LOWER algorithm.

Input: Task set A, computing server set K, revenue set P, transmission rate $r_{i,j}$ and computation capability $v_{i,j}$ from a_j to s_i; transmission rate R_i and computation capacity V_i; stack S_i and current solution $[\hat{x}]$; current computing server m
Output: Lower bound L and corresponding solution $[\tilde{x}]$
Steps:
1: $L = \sum_{i=1}^{m} \sum_{j \in Q_m} p_{i,j} \cdot \hat{x}_{i,j}$;
2: $N' = \{j \,|\hat{x}_{i,j} = 0, i = 1, ..., m\}$;
3: $\bar{N} = N' \backslash Q_m$;
4: $\bar{R} = R_i - \sum_{j \in Q_m} r_{m,j} \cdot \hat{x}_{m,j}$;
5: $\bar{V} = V_i - \sum_{j \in Q_m} v_{m,j} \cdot \hat{x}_{m,j}$;
6: $i = m$;
7: **while** $i \leq |S|$: **do**
8: Solve the two-dimensional single knapsack problem by allocating items in $\bar{N}$, with the resource constraints $\bar{R}$ and $\bar{V}$. Store the solution vector in row i of $[\tilde{x}]$.
9: $L = L + \bar{z}$.
10: $N' = N' \setminus \{j \,|\tilde{x}_{i,j} = 1\}$.
11: $\bar{N} = N'$.
12: $i = i + 1$.
13: $\bar{R} = R_i$, $\bar{V} = V_i$.
14: **end while**

TABLE V
THE UPPER ALGORITHM

Algorithm 2 The UPPER algorithm.

Input: Task set A, computing server set S, revenue set P, transmission rate $r_{i,j}$ and computation capability $v_{i,j}$ from a_j to s_i; transmission rate R_i and computation capacity V_i; stack Q_i and current solution $[\hat{x}]$; current computing server m
Output: Upper bound U_0
Steps:
1: $\bar{R} = R_m - \sum_{j \in Q_m} r_{m,j} \cdot \hat{x}_{m,j} + \sum_{i=m+1}^{|S|} R_i$;
2: $\bar{V} = V_m - \sum_{j \in Q_m} v_{m,j} \cdot \hat{x}_{m,j} + \sum_{i=m+1}^{|S|} V_i$;
3: Remove the classes from which a task has been chosen. That is, $\bar{N} = \{N_j \,|x'_{i,j}{=}0, \forall i \in N_j\}$;
4: Solve the MCKP by allocating items in $\bar{N}$, with the resource constraints $\bar{R}$ and $\bar{V}$.
5: $U = \sum_{i=1}^{m} \sum_{j \in Q_i} p_{i,j} \cdot \hat{x}_{i,j} + \bar{z}$.

$\hat{x}_{i,j}$ in stack Q_i and the heuristic solution $[\tilde{x}]$. If the optimal value z is equal to upper bound UB, the algorithm stops. If the optimal value z is equal to upper bound U_0, it means the heuristic solution has been completely traversed, and then, the algorithm goes backtrack. Otherwise, the branching procedure is conducted by iteratively choosing the allocation variable with the minimum index which equals to '1'. It pushes the task into the stack and calls the UPPER algorithm to compute a new upper bound U_0. Once U_0 is smaller than the current optimal value z, a backtrack process is called. Step 4 executes a backtrack process. It iteratively pops the task at the top of the stack and compute new upper bound U_0. If U_0 is larger than the current optimal value z, the LOWER algorithm is called again to derive a new heuristic solution.

TABLE VI
THE BOUND-AND-BOUND PROCDURE

Algorithm 3 The bound-and-bound procedure.

Input: Task set A, computing server set S, revenue set P, required transmission rate $r_{i,j}$ and computation capability $v_{i,j}$ from a_j to s_i; transmission rate R_i and computation capacity V_i; stack Q_i and current solution $[\hat{x}]$; m
Output: z and $[x]$
Steps:
1: *Step 1: Initialize*
2: **for** $i = 1, ..., |S|$ **do**
3: $Q_i = \emptyset$.
4: **for** $j = 1, ..., |A|$ **do**
5: $\hat{x}_{i,j} = 0$.
6: **end for**
7: **end for**
8: $z = 0$, $m = 1$.
9: Call UPPER and the surrogate relaxation to compute U_0 and U_1, and set $UB = \min\{U_1, U_0\}$.
10: *Step 2: Heuristic procedure*
11: Call LOWER to compute L and $[\tilde{x}]$.
12: **if** $L > z$ **then**
13: $z = L$;
14: **for** $i = 1, ..., |S|$ and $j = 1, ..., |A|$ **do**
15: $x_{i,j} = \hat{x}_{i,j}$.
16: **end for**
17: **for** $i = m, ..., |S|$ and $j = 1, ..., |A|$ **do**
18: **if** $\tilde{x}_{i,j} = 1$ **then**
19: $x_{i,j} = 1$.
20: **end if**
21: **end for**
22: Return if $z = UB$.
23: Goto Step 4 if $z = U_0$.
24: **end if**
25: *Step 3: Generate a new solution*
26: **while** $m \leq |S|$ **do**
27: $J = \{j \,|\tilde{x}_{m,j} = 1\}$;
28: **while** $J \neq \emptyset$ **do**
29: $j = \min\{l \,|l \in J\}$;
30: $J = J \backslash \{j\}$;
31: Push a_j on Q_m;
32: $\hat{x}_{m,j} = 1$;
33: Call UPPER to compute U_0;
34: Goto step 4 if $U_0 \leq z$.
35: **end while**
36: $m = m + 1$.
37: **end while**
38: $m = |S| - 1$.
39: *Step 4: Backtrack*
40: **while** $i \geq 0$ **do**
41: **while** $Q_m = \emptyset$ **do**
42: Let a_j be the item on the top of Q_i;
43: **if** $\hat{x}_{m,j} = 0$ **then**
44: Pop a_j from Q_i;
45: **else**
46: $\hat{x}_{m,j} = 0$;
47: Call UPPER to compute U_0 at current branch;
48: Goto step 2 if $U_0 \geq z$.
49: **end if**
50: **end while**
51: **end while**

REFERENCES

[1] Y. Gu, Z. Chang, M. Pan, L. Song, and Z. Han, "Joint radio and computational resource allocation in IoT fog computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 8, pp. 7475–7484, Aug. 2018.

[2] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[3] D. Chen, Y. Liu, B. Kim, J. Xie, C. S. Hong, and Z. Han, "Edge computing resources reservation in vehicular networks: A meta-learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5634–5646, May 2020.

[4] X. Zhang, J. Zhang, Z. Liu, Q. Cui, X. Tao, and S. Wang, "Mdp-based task offloading for vehicular edge computing under certain and uncertain transition probabilities," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3296–3309, Mar. 2020.

[5] Y. Wang *et al.*, "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, Jun. 2020.

[6] Y. J. Ku, P. H. Chiang, and S. Dey, "Real-time QoS optimization for vehicular edge computing with off-grid roadside units," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 11 975–11 991, Oct. 2020.
[7] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 9041–9052, Aug. 2020.
[8] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.
[9] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.
[10] Y. Sun, S. Zhou, and Z. Niu, "Distributed task replication for vehicular edge computing: Performance analysis and learning-based algorithm," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1138–1151, Feb. 2021.
[11] Z. Jiang, S. Zhou, X. Guo, and Z. Niu, "Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 93–107, Feb. 2018.
[12] M. Sookhak *et al.*, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Veh. Technol. Mag.*, vol. 12, no. 3, pp. 55–64, Sep. 2017.
[13] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.
[14] J. Wang, K. Liu, B. Li, T. Liu, R. Li, and Z. Han, "Delay-sensitive multi-period computation offloading with reliability guarantees in fog networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 9, pp. 2062–2075, Sep. 2020.
[15] S. K. Datta, J. Haerri, C. Bonnet, and R. F. D. Costa, "Vehicles as connected resources: Opportunities and challenges for the future," *IEEE Veh. Technol. Mag.*, vol. 12, no. 2, pp. 26–35, Jun. 2017.
[16] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.
[17] Z. Wang, Z. Zhong, and M. Ni, "Application-aware offloading policy using smdp in vehicular fog computing systems," in *Proc. IEEE Int. Conf. Commun. Workshops*, Kansas City, MO, USA, 2018, pp. 1–6.
[18] J. Wang, T. Liu, K. Liu, B. Kim, J. Xie, and Z. Han, "Computation offloading over fog and cloud using multi-dimensional multiple knapsack problem," in *Proc. IEEE Glob. Commun. Conf.*, Abu Dhabi, United Arab Emirates, 2018, pp. 1–7.
[19] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Trans. Mobile Comput.*, pp. 1–13, vol. 21, no. 2, pp. 598–611, Feb. 2022.
[20] R. Zhang, P. Cheng, Z. Chen, S. Liu, Y. Li, and B. Vucetic, "Online learning enabled task offloading for vehicular edge computing," *IEEE Wireless Commun. Lett.*, vol. 9, no. 7, pp. 928–932, Jul. 2020.
[21] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Commun.*, vol. 26, no. 1, pp. 87–93, Feb. 2019.
[22] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9637–9650, Oct. 2020.
[23] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE Int. Conf. Comput. Commun.*, San Francisco, CA, USA, 2016, pp. 1–9.
[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.
[25] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.
[26] S. Misra and N. Saha, "Detour: Dynamic task offloading in software-defined fog for iot applications," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1159–1166, May 2019.
[27] C. You and K. Huang, "Exploiting non-causal cpu-state information for energy-efficient mobile cooperative computing," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4104–4117, Jun. 2018.
[28] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[29] N. Cheng *et al.*, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.
[30] W. Zhan *et al.*, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, Jun. 2020.
[31] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma, and Z. Liu, "A novel cost optimization strategy for SDN-enabled UAV-assisted vehicular computation offloading," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3664–3674, 2021.
[32] H. Liao *et al.*, "Learning-based intent-aware task offloading for air-ground integrated vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5127–5139, Aug. 2021.
[33] H. Wu, F. Lyu, C. Zhou, J. Chen, L. Wang, and X. Shen, "Optimal uav caching and trajectory in aerial-assisted vehicular networks: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 12, pp. 2783–2797, Dec. 2020.
[34] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3113–3125, Apr. 2019.
[35] L. Pu, X. Chen, G. Mao, Q. Xie, and J. Xu, "Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 84–99, Feb. 2019.
[36] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Ave: Autonomous vehicular edge computing framework with aco-based scheduling," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10 660–10 675, Dec. 2017.
[37] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, Feb. 2019.
[38] Y. Sun, S. Zhou, and Z. Niu, "Distributed task replication for vehicular edge computing: Performance analysis and learning-based algorithm," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1138–1151, Feb. 2021.
[39] H. Sami, A. Mourad, and W. El-Hajj, "Vehicular-obus-as-on-demand-fogs: Resource and context aware deployment of containerized micro-services," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 778–790, Apr. 2020.
[40] X. Han *et al.*, "Reliability-aware joint optimization for cooperative vehicular communication and computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5437–5446, Aug. 2021.
[41] P. Dai, K. Hu, X. Wu, H. Xing, F. Teng, and Z. Yu, "A probabilistic approach for cooperative computation offloading in MEC-assisted vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: 10.1109/TITS.2020.3017172.
[42] F. Sun *et al.*, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11 049–11 061, Nov. 2018.
[43] Z. Wang, S. Zheng, Q. Ge, and K. Li, "Online offloading scheduling and resource allocation algorithms for vehicular edge computing system," *IEEE Access*, vol. 8, pp. 52 428–52 442, Mar. 2020.
[44] S. Huang, B. Lv, R. Wang, and K. Huang, "Scheduling for mobile edge computing with random user arrivalsan approximate MDP and reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7735–7750, Jul. 2020.
[45] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun.*, Atlanta, GA, USA, 2017, pp. 1–9.
[46] M. Xing, J. He, and L. Cai, "Maximum-utility scheduling for multimedia transmission in drive-thru internet," *IEEE Trans. Veh. Technol.*, vol. 65, no. 4, pp. 2649–2658, Apr. 2016.
[47] H. Zhou *et al.*, "Spatial coordinated medium sharing: Optimal access control management in drive-thru internet," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2673–2686, Oct. 2015.
[48] D. Committee *et al.*, "Dedicated short range communications (DSRC) message set dictionary," Society of Automotive Engineers, *Tech. Rep. J2735*, Nov. 2009.
[49] F. Lyu *et al.*, "Towards rear-end collision avoidance: Adaptive beaconing for connected vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 2, pp. 1248–1263, Feb. 2021.
[50] Y. L. Morgan, "Notes on DSRC & WAVE standards suite: Its architecture, design, and characteristics," *IEEE Commun. Surv. Tut.*, vol. 12, no. 4, pp. 504–518, May 2010.
[51] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018.

[52] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung, "Hybrid computation offloading in fog and cloud networks with non-orthogonal multiple access," in *Proc. IEEE Conf. Comput. Commun. Workshops*, Honolulu, HI, USA, 2018, pp. 154–159.
[53] L. Liu, Z. Chang, and X. Guo, "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1869–1879, Jun. 2018.
[54] M. Keller and H. Karl, "Response time-optimized distributed cloud resource allocation," in *Proc. ACM SIGCOMM Workshop Distrib. Cloud Comput.*, New York, NY, USA, 2014, Art. no. 47C52.
[55] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fibercwireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.
[56] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, "Optimal multiserver configuration for profit maximization in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1087–1096, Jun. 2013.
[57] S. Martello and P. Toth, "Knapsack problems: Algorithms and computer implementations," *Wiley-Interscience Ser. discrete Math. Optim.*, 1990.
[58] D. Pisinger and P. Toth, *Knapsack Problems*. US: Springer, 1999, pp. 299–428.
[59] U. P. H. Kellerer and D. Pisinger, *Knapsack Problems*. Berlin Heidelberg: Springer-Verlag, 2004.
[60] D. Chakrabarty, Y. Zhou, and R. Lukose, "Online knapsack problems," in *Proc. Workshop Internet Netw. Econ.*, Shanghai, China, 2008.
[61] R. Cai, Y. Feng, D. He, Y. Xu, Y. Zhang, and W. Xie, "A combined cable-connected RSU and uav-assisted RSU deployment strategy in V2I communication," in *Proc. IEEE Int. Conf. Commun.*, Dublin, Ireland, 2020, pp. 1–6.
[62] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, Jul. 2020.

Junhua Wang received the B.S. and Ph.D. degrees in computer science from Chongqing University, China, in 2014, and 2019, respectively. From 2017 to 2018, she was a Visiting Scholar with the University of Houston, USA. Since 2019, she has been working with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. Her research interests include mobile computing, vehicular ad-hoc networks, and wireless networks.

Kun Zhu (Member, IEEE) received the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2012. He was a Research Fellow with the Wireless Communications, Networks, and Services Research Group, University of Manitoba, Canada. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include resource allocation in 5G, wireless virtualization, and self-organizing networks. He was a TPC for several conferences and a reviewer for several journals.

Bing Chen received the B.S. and M.S. degrees in computer engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 1992 and 1995, respectively, and the Ph.D. degree from the College of Information Science and Technology, NUAA, China, in 2008. Since 1998, he has been with NUAA . He is currently a Professor with the Computer Science and Technology Department, NUAA. His main research interests include cloud computing, wireless communications, and cognitive radio networks.

Zhu Han (Fellow, IEEE) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1997, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, MD, USA, in 1999 and 2003, respectively.

From 2000 to 2002, he was an R&D Engineer of JDSU, Germantown, Maryland. From 2003 to 2006, he was a Research Associate with the University of Maryland. From 2006 to 2008, he was an Assistant Professor with Boise State University, Idaho. He is currently a John and Rebecca Moores Professor with the Electrical and Computer Engineering Department and with the Computer Science Department, the University of Houston, Texas. His research interests include wireless resource allocation and management, wireless communications and networking, game theory, big data analysis, security, and smart grid. Dr. Han was the recipient of NSF Career Award in 2010, Fred W. Ellersick Prize of the IEEE Communication Society in 2011, EURASIP Best Paper Award for the Journal on Advances in Signal Processing in 2015, IEEE Leonard G. Abraham Prize in the field of Communications Systems (Best Paper Award in IEEE JSAC) in 2016, and several best paper awards in IEEE conferences. Dr. Han was an IEEE Communications Society Distinguished Lecturer from 2015–2018, AAAS fellow since 2019 and ACM distinguished Member since 2019. Dr. Han is 1% highly cited Researcher since 2017 according to Web of Science. Dr. Han is also the winner of 2021 IEEE Kiyo Tomiyasu Award, for outstanding early to mid-career contributions to technologies holding the promise of innovative applications, with the following citation: "for contributions to game theory and distributed management of autonomous communication networks."