Combinatorial Optimization-based Task Allocation Mechanism for Vehicular Clouds

Joahannes B. D. da Costa*, Rodolfo I. Meneguette†, Denis Rosário‡, Leandro A. Villas*
*Institute of Computing (IC), University of Campinas (UNICAMP), Campinas, São Paulo, Brazil
†University of São Paulo (USP), São Carlos, São Paulo, Brazil
‡Federal University of Pará (UFPA), Belém, Pará, Brazil
Email: joahannes@lrc.ic.unicamp.br, meneguette@icmc.usp.br, denis@ufpa.br, leandro@ic.unicamp.br

***Abstract*—The automotive industry has been continuously investing in the modernization of the vehicles by the addition of more sensors and computational power. With this evolution, Intelligent Transportation Systems (ITS) make up a services framework that seeks to mitigate problems in the road sector. Many ITS services are facilitated by creating vehicular clouds (VCs) by using the communication capabilities of other vehicles to provide cloud services closer to vehicular applications. However, often the computational resources present in the vehicles are underutilized. For this reason, we propose in this work a mechanism that efficiently allocates computational tasks to be performed in VCs. Simulation results on a realistic mobility trace show that, with our mechanism, tasks are more allocated, the reward from allocating these tasks was higher, resource waste was minimized, and less CPU is used in the allocation processing. Also, the proposed mechanism is statistically close to a globally optimal solution.**

***Index Terms*—Vehicular cloud, Task allocation, ITS, VANETs**

I. Introduction

In recent years, the number of vehicles has grown significantly in the worldwide, contributing to the emergence of new research directions as the industry invests in incorporating more technological resources into vehicles [1]. In this way, vehicles could collect, transmit, and interpret information to help in data acquisition and decision making with the intent to help driver and devices to take action [2]. Vehicles play a significant role in Intelligent Transportation Systems (ITS). However, delay-intolerant requirements, such as safety applications require low latency, and also vehicular applications must improve the usage of network/computing resources.

In this context, Vehicular Cloud (VC) emerged as a promising area to provide cloud services closer to vehicular users to meet their needs regardless of vehicle location, improving road safety and ensuring intelligent urban traffic systems [3]. VC construction occurs by creating clusters with a set of vehicles sharing the same preferences, such as direction, path similarity, etc [4]. At any given moment, a vehicle could increase its capabilities by using resources available from the VC, while other vehicles lend their resources to the VC [5]. In this way, VC aggregates vehicle resources, such as communication, processing, and storage, from parked or moving vehicles to create a VC in a distributed fashion [6]. Hence, VC resources create a pool of services available to other vehicles [7], where a VC Controller deployed at the network edge to manage the task allocation and availability of resources.

The high vehicle mobility in both highway and urban environments makes the resource allocation in the VC environment an open issue, especially for processing tasks with delay-intolerant requirements [8]. Also, allocate tasks with high computational power requirements, such as traffic image processing and other multimedia applications, are challenging issues [9]. That is, VC's resources vary depending on the vehicular density variation. Thus, ensuring that the tasks are allocated and served in the VC regardless of their characteristics and requirements, is the main objective of studies in VC. In addition, it is important to consider an efficient task allocation mechanism to use VC resources efficiently, reducing the waste of computational resources, while provides computational power closer to vehicular applications.

In this paper, we propose a **C**ombinato**R**ial optimization-b**A**sed **T**ask all**O**cation mechani**S**m for VC, called **CRATOS**. It runs on VC Controller deployed at the network edge, such as a network controller, to coordinate multiple VCs. A vehicle requests resources to VC Controller, which is aware of resources available at each VC and task demands. In this way, CRATOS selects the optimum set of tasks to be allocated in real-time in each available VC based on combinatorial optimization as quickly as possible, and using the maximum computational resources as it can. CRATOS gives priority to allocate tasks in VCs with more available resources to maximize the fulfillment of demands in as few rounds as possible. Besides, we have implemented an Integer Linear Programming (ILP) formulation to show how close CRATOS is to the optimal global solution. The simulation results considering a realistic mobility trace shows the benefits of CRATOS mechanism, which offers 6.4% more tasks allocated, 46.69% higher allocation reward, 17.47% less wasted resources, and 30.67% less CPU utilization.

The rest of this paper is structured as follows. Section II discusses the related works on task allocation for VC. Section III introduces the system model, problem definition, and CRATOS operation. Section IV discusses the performance evaluation and results obtained. Finally, Section V presents the conclusions and future works.

II. Related Work

Pereira et al. [10] proposed a policy for allocation based on the method analytic hierarchy process to maximize resource availability in VCs for highway environments. Specifically,

IEEE 91st Vehicular Technology Conference (VTC-Spring 2020)

vehicles cooperate with the set of Fogs along the highway to create a pool of resources that will be made available to vehicular services. The use of the fog computing paradigm allows resources to be close to vehicles, allowing fog's resources to be added to the resources provided by vehicles, thus increasing the number of services that can be offered. However, this work does not consider only the vehicle resources for allocating services, in addition to considering a highway scenario that has more predictable mobility than an urban scenario.

Nabi et al. [11] presented a task allocation scheme for VCs that uses Knapsack Problem concepts. This scheme solves the allocation for a single task in polynomial time and provides a greedy solution for the same purpose. Besides, they extend the algorithm to solve the allocation problem for *n* tasks with lower bounds and Fractional Knapsack Problem. However, this scheme consider an offline environment in which the problem instance is entirely available and is known before the beginning of the simulation. In this sense, it requires prior knowledge of the tasks to be allocated, not operating satisfactorily in a real-time decision-making environment.

Wang et al. [12] proposed a task offloading algorithm for VCs, which builds on the concepts of the Multi-dimensional Multiple Knapsack Problem (MMKP). It considers that each user will pay for computing tasks according to their size, to maximize the total profits from computing offload from an infrastructure perspective. A modified Branch-and-Bound algorithm is proposed to obtain the ideal solution with a Greedy heuristic method to get approximate performance with lower computational overhead. However, these algorithms are computationally costly because they compute the optimal solution for each knapsack in one step and only then use this information to allocate tasks in different clouds.

Hattab et al. [13] introduced a polynomial time complexity algorithm for task allocation in VCs with different computational resources. First, the algorithm classifies the tasks according to the ratio between completion time and wait time. Afterwards, it selects a subset of tasks with the lowest proportion and then solves a sequence of Linear Programs. They formulate the bottleneck assignment problem, where the goal is to minimize the completion time of the allocated tasks in the available VCs. However, this work does not consider the mobility of vehicles for VC formation, *i.e.*, VCs are stationary, and the proposed algorithm considers only one VC.

Based on our analysis of the state-of-the-art, we conclude that existing works do not consider that tasks arrive in the system dynamically and independently to allocate the tasks in real-time. Besides, it is essential to consider VCs managed by a controller deployed closer to the user, such as in a RSU.

III. CRATOS

This section describes the task allocation mechanism, where we consider a scenario composed of multiple VCs coordinated by a VC Controller, which runs CRATOS for task allocation. CRATOS considers a combinatorial optimization approach for selecting the optimum set of tasks to be allocated in real-time in the available VCs.

A. Overview

Figure 1 shows a task allocation scenario in a VC environment. We consider a scenario composed of x moving vehicles $u_i \in U$, and each vehicle has an individual identity ($i \in [1, x]$). These vehicles can be represented in a dynamic graph $G(V, E)$, where the vertices $U = u_1, \ldots, u_x$ represent a finite set of vehicles, and edges $E = e_1, \ldots, e_x$ build a finite set of asymmetric wireless links between them. Each vehicle can communicate with the neighbor vehicle (V2V) and/or with infrastructure (V2I).

At any moment, a given vehicle u_i needs to process some data, but its computational resources do not support to process such a task. In this sense, it sends a request message to VC Controller to find neighbor vehicles $N(u_i) \subset U$ that could lend it their resources to perform its task [7]. A group of vehicles sharing the same preferences, such as direction and path similarity, create a VC with computing, sensing, and physical resources to be coordinated by the VC Controller [3]. VC formation does not depend on a specific technique for clustering since the only requirement is that VCs must be identified and provided. In this way, we consider a well-known clustering technique, named Density-Based Spatial Clustering of Application with Noise (DBSCAN) [14].

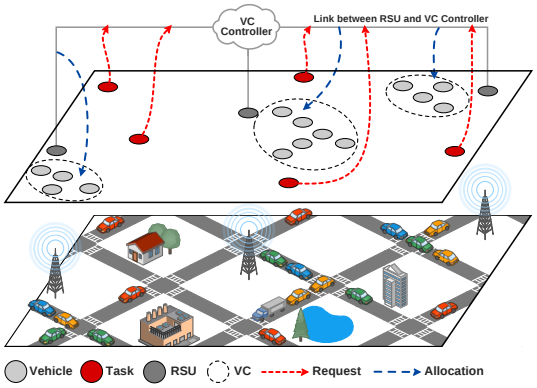


Fig. 1. VC environment.

We denote each VC as $v_j \in V = \{v_1, \ldots, v_m\}$, which consists of a subset of vehicles $V \subset U$ capable of sharing up to 3 of their computational resources ω_i, such as i) bandwidth, ii) processing and/or iii) storage. The VC controller is deployed at the network edge, such as network controller, connected to different Road Side Unit (RSU) in order to coordinate multiple VCs. As soon as given a vehicle u_i requests resources to allocate its task, the VC Controller must be aware of who has sufficient resources to allocate this task. In this sense, the VC Controller must collect vehicle information in real-time, such as available resources, speed, direction, and GPS coordinates using V2I communication to maintain knowledge about the vehicular environment. Hence, VC Controller considers CRATOS to allocate the task (blue arrow) to any available VCs, which can perform the task from the vehicle u_i that requested resource (red arrow). CRATOS must perform task allocation as quickly as possible and using

maximum computational resources as it can, since the waste of resources must be minimized. It is important to mention that more than one task should be allocated to the same VCs if this is possible.

B. Problem Definition

We describe each task $t_k \in T = \{t_1, \ldots, t_n\}$, and can be represented as a 3-uple $\{id_k, w_k, g_k\}$, where id_k means the task identification ($k \in [1, n]$), w_k represents the amount of resources to be allocated, and g_k denotes the allocation reward. In this way, the allocation mechanism aims to optimum allocate such tasks to be processed in a VC without wast resources and allocating more task as possible.

The Vehicular Cloud Task Allocation Problem (VCTAP) can be modeled as a 0/1 Knapsack Problem (KP) because of its similar characteristics. In KP, there are a items with weight b and value-added c, and a Z capacity knapsack. The KP aims to maximize the value of items stored in the knapsack by not exceeding its capacity while considering that an item can only be stored once (0 for unallocated and 1 for allocated). KP is a combinatorial optimization problem and has no known resolution in polynomial time. Thus, KP is a $\mathcal{NP}$-hard problem with $\mathcal{O}(aZ)$ complexity, where a being the number of items and Z the capacity of the knapsack [15].

In this context, we define each task t_k as a knapsack item, and each VC v_j as the knapsack. The total amount of resource Ω_j of each VC v_j is the sum of the shared resources ω_i from each vehicle u_i belonging to a given VC v_j, as shown Eq. 1.

$$\Omega_j = \sum_{i=2}^{u} \omega_i, \forall u_i \in v_j \tag{1}$$

The reward g_k for allocating a task increases uniformly at twice its weight w_k. In this way, we give higher rewards to tasks with high weight, *i.e.*, tasks that require higher computational power to be solved. CRATOS considers a combinatorial optimization problem to maximize the reward for allocating tasks, while the waste of vehicular computational resources is minimized. The VCTAP is formally defined as:

$$\begin{aligned} &\text{maximize} \sum_{k=1}^{n} g_k t_k \\ &\text{subject to} \sum_{k=1}^{n} w_k t_k \leq \Omega_j, \text{ and } t_k \in \{0, 1\} \end{aligned} \tag{2}$$

C. Proposal for Efficient Task Allocation

VCTAP resolution requires minimal computational power use in a sufficiently short time to meet the critical requirements of sure ITS applications. As discussed, the problem of allocating tasks in VCs is $\mathcal{NP}$-hard [9]. We can use a dynamic programming approach to solve the VCTAP in pseudo-polynomial time with $\mathcal{O}(n\Omega)$ complexity, where n is the number of tasks and Ω the capacity of a given VC v_j. However, we consider that tasks should be efficiently allocated to more than one VCs if any, and adding more VCs to the problem increases the complexity of the problem.

Dynamic programming considering multiple knapsacks, known as 0/1 Multiple Knapsack Problem (MKP), is impractical because increasing its complexity. On the other hand, other solutions seek to list all possible possibilities and cut the decision tree based on relaxations, *i.e.*, defining upper and lower bounds. In this context, CRATOS considers a dynamic programming algorithm to solve traditional KP but considering multiple VCs, where its complexity becomes $\mathcal{O}(\max\{m\}n\Omega)$ in the worst case, with m being number of VCs.

Algorithm 1 describes the main operations for CRATOS allocate tasks in a VC environment. In this sense, the VC controller must provide to CRATOS the set of VC V and set of task T, which gives as an output the optimal task allocation $\mathbb{S}$ for each VC v_j. First, as long as unallocated tasks exist, the system will operate (Line 5). As soon as there are no available resources at the set of VC V, the system will stop as there are no more available resources to allocate (Line 15). Otherwise, CRATOS takes a given VC v_j with more available resources from the set of VC V. In addition, CRATOS call the procedure KNAPSACKPROBLEM (*i.e.*, traditional KP) giving the set of task T and the selected VC v_j, which returns tasks that were allocated in the selected VC v_j (Lines 7-9). As tasks are allocated, they are removed from the set of task T and added to the allocated task set $\mathbb{S}$. Tasks that remain in the set of task T will be reallocated in the next round (Lines 10-11). The algorithm stops if all tasks have been allocated in this round (Line 13).

Algorithm 1: CRATOS

```
   Input: task set T and VC set V
   Output: allocated tasks 𝕊
1  begin
2      T ← {1, ..., n}
3      V ← {1, ..., m}
4      t ← ⟨t_{id_k}, t_{w_k}, t_{g_k}⟩ ∀t ∈ T
5      while T ≠ ∅ do
6          if V ≠ ∅ then
7              v ← max{V}
8              V ← V \ {v}
9              S' ← KnapsackProblem({T, v})
10             T ← T \ S'
11             𝕊.pushback(S')
12             if |T| = 0 then
13                 break          ▷ all tasks were allocated
14         else
15             break                      ▷ without resources
16     return 𝕊
```

IV. Evaluation

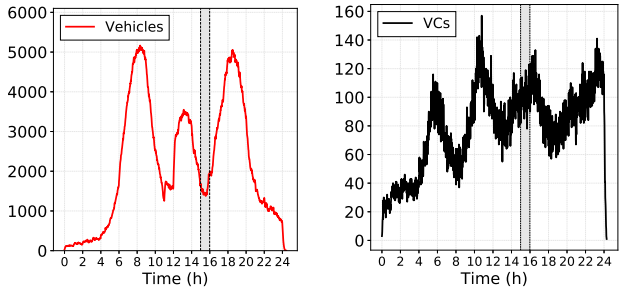
This section describes methodology and metrics used to evaluate CRATOS performance in terms of allocation reward, tasks allocated (called of services served), resource utilization, and CPU time compared to greedy and an ILP formulation.

A. Scenario description and Methodology

We implemented the task allocation in the Simulation of Urban MObility (SUMO) 0.29.0. The algorithms (CRATOS and GREEDY) were implemented in Python and connected to

SUMO through the TraCI[1] interface. We considered a real mobility trace from Luxembourg SUMO Traffic (LuST) [16], which provides 24 hours of mobility simulation with 5.000 vehicles at its peak times for the city of Luxembourg using the traffic simulator SUMO. Results present values with a confidence interval of 95%.

For this work, we consider only a period with a lower vehicular density from LuST, *i.e.*, from 3pm to 4pm, as indicated in Figure 2(a). In this way, we demonstrate that the CRATOS mechanism operates satisfactorily with fewer available resources. We consider each vehicle's communication range of 100m, and a minimum number of members per VC has been set to 2. Based on such values, Figure 2(b) shows the number of VCs identified in the trace, which follows the vehicular density of the scenario, but they are inversely proportional. According to vehicular density increases, the higher the chance of vehicles joining the clusters, *i.e.*, VCs.



(a) Number of vehicles (b) Number of VCs identified

Fig. 2. Data extracted from the LuST scenario.

The deadline (system runtime) for each task is equaled to 1, *i.e.*, once the task is allocated, its success in the allocation is already accounted. The clustering interval was 60 seconds. Tasks are generated after the cluster formation, and a task allocation mechanism allocates them based on its behavior. The tasks arrival rate follows a Poisson distribution with an average of $\lambda = 25$, as their occurrences are independent of each other [17]. In addition, the weight w of each task is defined as a uniform distribution ranging from $[1, \mu]$, where $\mu = \{300\}$ and the allocation reward g is defined between $[2, (w \times 2)]$, as discussed in Subsection III-B. We consider that vehicles make available resources ranging between 1 and 3, and average tasks weight equals to 300. In this way, we can get an idea of the impact that sharing different resource units employ on the system as a whole.

We considered three allocation mechanism to analyze their performance to allocate tasks in a VC environment, namely CRATOS, GREEDY, and ILP. GREEDY implementation is based on Nabi et al. [11], which is a greedy strategy that orders tasks in non-increasing weight, and allocates tasks based on that classification considering all available VCs. CRATOS considers a combinatorial optimization-based task allocation, as introduced in Section III. We also implemented a formulation of ILP to solver the 0/1 Multiple Knapsack Problem (MKP) [15]. This ILP provides the global optimum solution for the n tasks and m VCs, enabling to assess how close CRATOS and GREEDY are to the optimal global solution. We implemented ILP on solver Gurobi[2], 9.0 version. The formalization of ILP is similar to the formulation of traditional KP (Eq. 2) and can be defined based on Eq. 3. However, the assignment of items considers all knapsacks available in the same round.

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{m}\sum_{k=1}^{n} g_k t_k \\ &\text{subject to} \sum_{k=1}^{n} w_k t_k \leq \Omega_m, \text{ and } t_k \in \{0,1\} \end{aligned} \tag{3}$$

We consider the following metrics to evaluate the performance of task allocation mechanisms: i) *allocation reward* means how much reward was obtained for allocating tasks; ii) *services served* refers to the ratio of successfully allocated tasks; iii) *resource utilization* denotes how much VC resources were used; and iv) *CPU time* is the time spent by the allocation mechanism to select on which VC to allocate tasks.

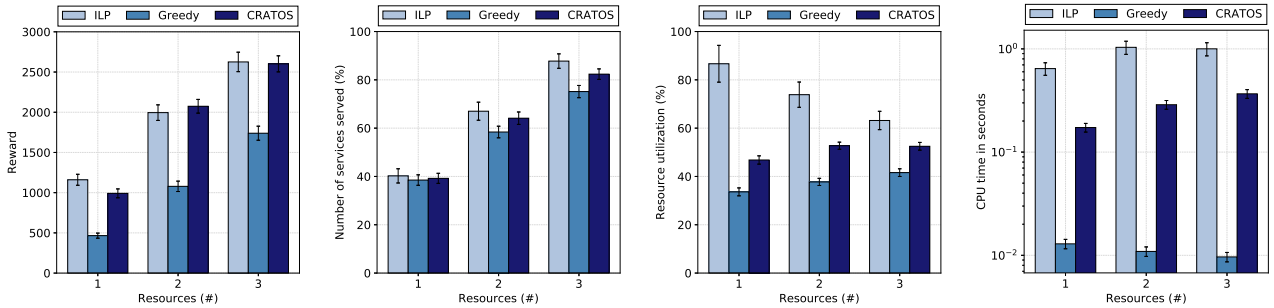
B. Results

Figure 3(a) shows reward per task allocation for the evaluated mechanisms. We can see that CRATOS has greater allocation reward compared to GREEDY approach. This is because CRATOS considers all available VCs and selects the one with the highest capacity to allocate more tasks per round and, consequently, it increases the average reward. GREEDY performs poorly, even when making the greedy choice based on the value of tasks, since it ends as soon as the task value from the current task is greater than the next task. On the other hand, CRATOS checks the weight and task value at the same time, and thus it finds an optimal solution for one VC at a time. However, as we can see, the ILP manages to obtain a better reward for allocation, since it combines all the possibilities given the n tasks and m VCs and always consider the best among the solutions. In general, the CRATOS employed improvement of 46.69% over GREEDY, and remains very close to the ILP, about 96.17%, which is the global optimum.

Figure 3(b) displays the rate of tasks that were successfully allocated for the mechanisms with different amounts of resources. As we can see, CRATOS can allocate more tasks in all scenarios compared to GREEDY. All the evaluated mechanisms behave similarly for vehicles sharing 1 resource unit. This is because as the number of resources is low to allocate heavy tasks, selecting the largest VCs first ensures that at least part of these tasks will be allocated without any criteria besides observing the size of the VCs. However, as soon as more resources are made available by vehicles in VCs, ILP stands out for having more VCs options to observe during the process of building the optimal solution. In VCs with greater availability of resources, CRATOS allocates 80% of the tasks.

[1]http://sumo.dlr.de/docs/TraCI.html

[2]http://www.gurobi.com



(a) Allocation reward (b) Services served (c) Resource utilization (d) CPU time

Fig. 3. Simulation results for different number of resources.

CRATOS employed improvement of 6.4% over GREEDY, and proved to be close to 95.82% of the ILP formulation.

Figure 3(c) presents the usage of available resources in the VCs, which is essential to show that vehicular applications can better exploit previously underutilized resources. As can be seen, CRATOS makes better use of available resources compared to GREEDY, since it keeps operating while tasks are being performed and while there are computable resources in VCs that can be allocated optimally. GREEDY is not concerned with using resources efficiently because it is primarily concerned with the reward of the task. Thus, CRATOS decreases wasted resources in 17.47% compared to GREEDY, and remains close to the ILP, about 69.49%.

Figure 3(d) shows the average CPU usage, where a longer CPU time potentially introduces latency to the system, consequently degrading its overall performance. For this evaluation, we counted the amount of time spent in each allocation step using an Intel(R) Core(TM) i7-8565U CPU (8×1.80 GHz) with Linux 64bits. ILP uses more processing resources to compute the global optimal task allocation solution. Next, CRATOS is 28.65% more computationally expensive than GREEDY, but even so, it is 30.67% as cheap as ILP.

In general terms, CRATOS shows itself close to the ILP in terms of allocated tasks and the reward obtained for these allocations, making better use of computational resources concerning GREEDY, spending less computational power required by ILP because of its complexity.

V. CONCLUSION

In this paper, we proposed a task allocation mechanism called CRATOS, which is based on combinatorial optimization for efficient task allocation in VC environment to make the most of the computational resources available and enable cloud services closer to vehicular users. Simulation results show that CRATOS fulfills its objectives to minimize the waste of resources while allocating a considerable number of tasks compared to the GREEDY approach, and it is close to the ILP. The future works include spatiotemporal analysis to estimate the duration of the VCs, thereby increasing the availability of resources for task allocation with more challenging deadlines. Also, the measurement of network metrics about the maintenance of knowledge by the VC Controller.

REFERENCES

[1] Qualcomm. (2018) Connecting vehicles to everything. [Online]. Available: https://www.qualcomm.com/invention/5g/cellular-v2x

[2] F. Dressler, G. S. Pannu, F. Hagenauer, M. Gerla, T. Higuchi, and O. Altintas, "Virtual edge computing using vehicular micro clouds," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 537–541.

[3] A. Boukerche and E. Robson, "Vehicular cloud computing: Architectures, applications, and mobility," *Computer Networks*, 2018.

[4] F. Hagenauer, C. Sommer, T. Higuchi, O. Altintas, and F. Dressler, "Vehicular micro cloud in action: On gateway selection and gateway handovers," *Ad Hoc Networks*, vol. 78, pp. 73–83, 2018.

[5] G. S. Pannu, F. Hagenauer, T. Higuchi, O. Altintas, and F. Dressler, "Keeping data alive: Communication across vehicular micro clouds," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[6] A. Thakur and R. Malekian, "Fog computing for detecting vehicular congestion, an internet of vehicles based approach: A review," *Intelligent Transportation Systems Magazine*, vol. 11, no. 2, pp. 8–16, 2019.

[7] F. Hagenauer, T. Higuchi, O. Altintas, and F. Dressler, "Efficient data handling in vehicular micro clouds," *Ad Hoc Networks*, vol. 91, p. 101871, 2019.

[8] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions," *Wireless Communications and Mobile Computing*, 2019.

[9] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Transactions on Vehicular Technology*, 2019.

[10] R. Pereira, D. Lieira, M. da Silva, A. Pimenta, J. B. D. da Costa, D. L. Rosario, and R. I. Meneguette, "A novel fog-based resource allocation policy for vehicular clouds in the highway environment," in *Proceedingsof the IEEE 11th Latin-American Conference on Communications (LATINCOM)*. IEEE, 2019, pp. 1–6.

[11] M. Nabi, R. Benkoczi, S. Abdelhamid, and H. S. Hassanein, "Resource assignment in vehicular clouds," in *2017 IEEE International Conference on Communications (ICC)*. Ieee, 2017, pp. 1–6.

[12] 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 *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[13] G. Hattab, S. Ucar, T. Higuchi, O. Altintas, F. Dressler, and D. Cabric, "Optimized assignment of computational tasks in vehicular micro clouds," in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*. ACM, 2019, pp. 1–6.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[15] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons Ltd., 1990.

[16] L. Codeca, R. Frank, and T. Engel, "Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research," in *IEEE Vehicular Networking Conference (VNC)*. IEEE, 2015, pp. 1–8.

[17] A. J. Kadhim and S. A. H. Seno, "Maximizing the utilization of fog computing in internet of vehicle using sdn," *IEEE Communications Letters*, vol. 23, no. 1, pp. 140–143, 2019.