UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALISSON BARBOSA DE SOUZA

A CONTEXT-ORIENTED FRAMEWORK AND DECISION ALGORITHMS FOR COMPUTATION OFFLOADING IN VEHICULAR EDGE COMPUTING

FORTALEZA

2021

ALISSON BARBOSA DE SOUZA

A CONTEXT-ORIENTED FRAMEWORK AND DECISION ALGORITHMS FOR COMPUTATION OFFLOADING IN VEHICULAR EDGE COMPUTING

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito para a obtenção do Título de Doutor em Ciência da Computação. Área de Concentração: Ciência da Computação.

Orientador: Prof. Dr. José Neuman de Souza

Coorientador: Prof. Dr. Paulo Antonio Leal Rego

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S713c Souza, Alisson Barbosa de.
A context-oriented framework and decision algorithms for computation offloading in vehicular edge computing / Alisson Barbosa de Souza. – 2021.
136 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2021.
Orientação: Prof. Dr. José Neuman de Souza.
Coorientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Computation Offloading. 2. Vehicular Edge Computing. 3. Contextual Information. 4. Artifical Bee Colony. 5. Vehicular Networks. I. Título.

CDD 005

ALISSON BARBOSA DE SOUZA

A CONTEXT-ORIENTED FRAMEWORK AND DECISION ALGORITHMS FOR COMPUTATION OFFLOADING IN VEHICULAR EDGE COMPUTING

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito para a obtenção do Título de Doutor em Ciência da Computação. Área de Concentração: Ciência da Computação.

Aprovada em: 15 de Julho de 2021

BANCA EXAMINADORA

Prof. Dr. José Neuman de Souza (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Antonio Leal Rego
Universidade Federal do Ceará (UFC)

Prof. Dr. Emanuel Bezerra Rodrigues
Universidade Federal do Ceará (UFC)

Prof. Dr. Luís Henrique Maciel Kosmalski Costa
Universidade Federal do Rio de Janeiro (UFRJ)

Profa. Dra. Susana Isabel Barreto de Miranda Sargento
Universidade de Aveiro (UA), Portugal

ACKNOWLEDGEMENTS

First, I would like to offer thanks to almighty God, Jehovah, for his loyal love, for life, for hope, in short, for everything.

I express my deepest gratitude to my parents, Valter and Ana, for the love, advice, and support at all times. This work would not have been possible without them. I also thank my brother and my sister-in-law, Gleidson and Carolina, for helping me at various times with support and incentives.

I am grateful to my advisor, Prof. Dr. José Neuman de Souza, for guidance, help, advice, and trust. I am also thankful to my co-advisor, Prof. Dr. Paulo Antonio Leal Rego, for countless conversations, guidance, and support. Their dedication and effort were fundamental to the accomplishment of this work.

I am thankful to my friend, Prof. Dr. Tiago Carneiro, for the support, incentives, tips, and conversations. I am also grateful to the colleague of the GREat laboratory at the Federal University of Ceará, Paulo Henrique Rocha, for having helped me at various times with the computational execution of the massive amount of experiments.

My thanks also go to everyone who helped me at some point during my PhD with actions, incentives, or suggestions, such as: Dr. Marcelo Esmeraldo Holanda, Prof. Dr. Alberto Sampaio Lima, Prof. Dr. Joaquim Celestino Júnior, Profa. Dra. Michele Nogueira Lima, Profa. Dra. Susana Isabel Barreto de Miranda Sargento, Prof. Dr. Manuel Gonçalves da Silva Neto, Prof. Dr. Emanuel Bezerra Rodrigues, Prof. Dr. Luís Henrique Maciel Kosmalski Costa, Prof. Dr. Jeandro de Mesquita Bezerra, and Prof. Dr. André Jalles Monteiro.

I would like to thank the Federal University of Ceará for giving me the opportunity to grow as a person, student, and professional. I also thank my colleagues from the Quixadá Campus for their support and for allowing me to obtain the leave to complete my PhD.

Finally, I would like to thank all the people who contributed to the realization of this work in one way or another.

“A spirit is manifest in the laws of the Universe — a spirit vastly superior to that of man, and one in the face of which we with our modest powers must feel humble.”

(Albert Einstein)

RESUMO

Veículos autônomos e aplicações veiculares complexas têm se tornado cada vez mais populares e requerem massivos recursos computacionais. Apesar de os veículos estarem se tornando mais conectados e inteligentes, eles ainda não possuem poder computacional suficiente para atender a essas demandas de modo satisfatório. Uma opção para lidar com esse desafio é permitir que recursos computacionais de veículos vizinhos e servidores de borda acoplados às estações base sejam utilizados através de sistemas de computação de borda veicular. Então, os veículos podem enviar tarefas, ou partes menores de aplicações, para esses servidores remotos através da técnica de *offloading* computacional. Nessa técnica, tais servidores executam as tarefas e retornam o resultado do processamento para o veículo inicial. Embora essa técnica vise diminuir o tempo de execução de aplicações, realizá-la em cenários veiculares é desafiador devido ao rápido movimento dos nós da rede e às frequentes desconexões. Em tais casos, informações contextuais que caracterizam a situação de dispositivos de redes e veículos ajudam a lidar com esses desafios por auxiliar processos de decisão de *offloading* a entregar melhores resultados. Assim, nós propomos um *framework* orientado a contexto e algoritmos de atribuição de tarefas para reduzir o tempo de execução de aplicações veiculares de forma confiável através de *offloading* computacional em sistemas de computação de borda veicular. O *framework* gerencia todas as etapas do processo de *offloading* e provê um mecanismo de recuperação de falhas. O módulo principal desse *framework* permite que os algoritmos propostos façam a atribuição das tarefas de aplicações para diferentes servidores, usando parâmetros contextuais e redes WAVE e 5G. Os resultados dos experimentos mostram que nossas soluções podem reduzir significativamente o tempo de execução de aplicações veiculares. Baseado na metaheurística colônia artificial de abelhas, nosso melhor algoritmo consegue que essa redução média atinja até 75,6 % se comparado à execução local e até 57,9 % se comparado a algoritmos da literatura, com até 0,0 % de falhas. Esses resultados mostram que as soluções propostas são uma alternativa promissora para viabilizar a execução de complexas aplicações veiculares.

Palavras-chave: *Offloading* Computacional. Redes Veiculares. Computação de Borda Veicular. Informação Contextual. WAVE. 5G. Colônia Artifical de Abelhas.

ABSTRACT

Autonomous vehicles and complex vehicular applications have become increasingly popular and require massive computational resources. Although vehicles are becoming more connected and intelligent, they still do not have enough computation power to satisfy these demands satisfactorily. One option to deal with this challenge is to allow computing resources from neighboring vehicles and edge servers coupled to base stations to be used through vehicular edge computing systems. Then, vehicles can send tasks, or smaller parts of applications, to these remote servers through the computation offloading technique. In this technique, such servers execute the tasks and return the processing result to the initial vehicle. Although this technique aims to reduce application execution time, performing it in vehicular scenarios is challenging due to the fast movement of network nodes and the frequent disconnections. In such cases, contextual information that characterizes the situation of network devices and vehicles helps to deal with these challenges by assisting offloading decision processes in delivering better results. Thus, we propose a context-oriented framework and task assignment algorithms to reduce the execution time of vehicular applications reliably through computation offloading in vehicular edge computing systems. The framework manages all stages of the offloading process and provides a failure recovery mechanism. The main module of this framework allows the proposed algorithms to assign application tasks to different servers, using contextual parameters and WAVE and 5G networks. Experimental results show that our solutions can significantly reduce the execution time of vehicular applications. Based on the artificial bee colony metaheuristic, our best algorithm achieves that this average reduction reaches up to 75.6 % compared to local execution and up to 57.9 % compared to literature algorithms, with up to 0.0 % of failures. These results show that the proposed solutions are a promising alternative to enable the execution of complex vehicular applications.

Keywords: Computation Offloading. Vehicular Networks. Vehicular Edge Computing. Contextual Information. WAVE. 5G. Artifical Bee Colony.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

4G	Fourth-Generation Cellular Mobile Networks
5G	Fifth-Generation Cellular Mobile Networks
ABC	Artificial Bee Colony
ALPR	Automatic License Plate Recognition
BCV	ABC for Computation Offloading in VEC
BS	Base Station
C-V2X	Cellular Vehicle to Everything
CV	Combustion-powered Vehicle
CVM	Combustion-powered Vehicle in Motion
CVO	Combustion-powered Vehicle Off
DN	Data Network
DSRC	Dedicated Short Range Communications
EV	Electric Vehicle
FIFO	First In, First Out
GCF	Greedy for CPU Free
GTT	Greedy Task by Task
HVC	Hybrid Vehicular edge Cloud
ITS	Intelligent Transportation Systems
MAC	Medium Access Control
mmWave	Millimeter Wave
NP	Non-Deterministic Polynomial-Time
OBU	On-Board Unit

PDU	Protocol Data Unit
RSU	Roadside Unit
UE	User Equipment
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VANET	Vehicular Ad Hoc Network
VC	Vehicular Cloud
VEC	Vehicular Edge Computing
WAVE	Wireless Access in Vehicular Environments

CONTENTS

1 INTRODUCTION

This chapter presents the contextualization of the problem addressed in this thesis in Section 1.1 and the statement of this problem in Section 1.2. Then, Sections 1.3 and 1.4 describe the research questions and the objectives. Contributions and publications are presented in Sections 1.6 and 1.7. Finally, Section 1.8 of this chapter presents the structural organization of this thesis.

1.1 Contextualization

The automobile industry has been one of the main sectors of the economy for over a century, and the number of vehicles produced continues to increase (ZHANG; LETAIEF, 2019). According to the latest report by the World Health Organization (WHO), there are more than two billion vehicles registered worldwide (WORLD HEALTH ORGANIZATION, 2020). In addition to quantity, the quality of vehicles has also demanded significant efforts to provide more comfort, safety, convenience, and efficiency for people (ZHANG; LETAIEF, 2019).

In this sense, different systems and technologies have been incorporated into vehicles. We can highlight the Wireless Access in Vehicular Environments (WAVE) architecture (JIANG; DELGROSSI, 2008) and the Fifth-Generation cellular mobile networks (5G) (MEZZAVILLA *et al.*, 2018; STORCK; DUARTE-FIGUEIREDO, 2019) among these technologies. They enable the formation of Vehicular Ad Hoc Networks (VANETs) that are used to provide connectivity to vehicles through Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communications (WEVERS; LU, 2017; AL-SULTAN *et al.*, 2014). Through these connections, vehicles can send collision, accident, and overtaking alert messages and improve road safety (ZHANG; LETAIEF, 2019). These connections can also make in-vehicle travel more enjoyable for drivers and passengers by allowing Internet access to vehicular applications (AL-SULTAN *et al.*, 2014).

In addition to advances in communications, vehicles have also evolved in intelligence through computing capabilities, cameras, embedded systems, sensors, and satellite navigation systems (ZHANG; LETAIEF, 2019). However, the advent of autonomous vehicles and new and popular applications such as augmented reality, automatic object recognition, and real-time video surveillance demand massive computing resources to deal with complicated data processing and critical latency requirements (BOUKERCHE; SOTORO, 2020; LIU *et al.*, 2020). The delay in executing one of these applications can compromise its usefulness, data validity, and even the

safety of people inside the vehicle (SOUZA *et al.*, 2020). Unfortunately, despite technological advances, vehicles do not yet have sufficient on-board computing resources (mainly proper CPUs) to handle all the vehicular applications requirements in a feasible time. Even if more powerful processors were installed in the vehicles, this could compromise their energy and displacement efficiency (ZHANG; LETAIEF, 2019).

One manner to assist vehicles with latency and processing requirements is the Vehicular Edge Computing (VEC) system. In this system, computational processing can be done on Vehicular Clouds (VCs) or edge servers, avoiding the excessive latency of sending data for processing on traditional cloud servers. VCs are a pool of computational resources of two or more vehicles, stationary or in motion, which can be dynamically coordinated to offer services on demand, through V2V connections and on-board units (OBUs), as in the cloud computing model. Although VCs have less latency in communication and operate in scenarios without infrastructure, they do not present great computational power. Another option is to use edge servers coupled with Roadside Units (RSUs) or base stations (BSs) through V2I connections. These servers are deployed close to streets and roads by service providers. Even though they have a little more communication latency and can be quite requested by network devices, these servers generally have greater computational capacity than vehicular clouds (SOUZA *et al.*, 2020).

One way to take advantage of these available computing resources is to apply the computation offloading technique, also called task offloading. This technique offloads smaller parts or tasks of an application to remote devices or servers, which are vehicles or edge servers in VEC systems. Then, these servers process the tasks and return the result to the client vehicle. Thus, computation offloading is used to improve applications' execution time and decrease the overload of processing (REGO *et al.*, 2017; XU *et al.*, 2018).

Nevertheless, it is a challenge to carry out computation offloading in VEC systems. Servers chosen to process tasks may already be overloaded. Vehicular networks also lack a central point to coordinate computing resources and message exchanges. Furthermore, the dynamic nature and the rapid movement of nodes in these networks lead to frequent disconnections and offloading failures (AL-SULTAN *et al.*, 2014). These issues can increase latency or interrupt vehicular applications' execution, reducing the offloading technique's effectiveness.

1.2 Problem Statement

Thus, the problem addressed in this thesis can be summarized as:

How to minimize the execution time of vehicular applications reliably through computation offloading in VEC systems?

However, for the computation offloading process to be efficient, some steps need to be supported and managed. Preliminary steps involve discovering computing resources and gathering contextual information. Through these steps, the client vehicle becomes aware of which servers are available and their contextual information. This information consists of descriptive parameters of location, direction, speed, energy, bandwidth and range of the communication technology used, and CPU availability and capacity. With this information gathered, the client vehicle can proceed to the most important step and the main part of the problem: the task distribution.

Deciding how to distribute the tasks is also the most complex step. In fact, finding the optimal way to assign computation tasks to different servers for maximum reduction in execution time of vehicular applications is a Non-Deterministic Polynomial-Time (NP)-hard problem. As such, there is no exact polynomial-time solution for this problem (ZHU *et al.*, 2018; FENG *et al.*, 2017). It is necessary to consider several contextual parameters to find an approximate solution. As each task and server have different requirements and characteristics, just a small change in the assignment process can significantly impact execution time results. Thus, this problem is also associated with solving the following questions about computation tasks: "where to send?", "which tasks to send?", "how to send?" (or what technology to use for sending), and "when to send?".

In addition, the entire computation offloading process must be reliable, i.e., offloading failures must be avoided, minimized, or handled. For example, assume two vehicles in opposite directions. One of them needs to offload computation tasks to reduce the execution time of one of their applications. However, suppose it offloads a task to the other vehicle. In that case, the latter can leave the first vehicle's range without returning the task or its processing result, generating a failure. Therefore, a reliable computation offloading solution must calculate the risks before offloading tasks to other devices and provide recovery mechanisms if any failure occurs.

1.3 Research Questions

As described in the previous section, the problem addressed in this thesis involves the reliable use of computation offloading in VEC systems to reduce the execution time of vehicular applications. Thus, the main Research Questions (RQs) of this thesis are:

- **RQ1: How to provide support and management for all stages of computation offloading processes?**
- **RQ2: How to assign computation tasks to different servers to reduce vehicular applications' execution time in VEC systems?**
- **RQ3: How to avoid offloading failures and, ultimately, recover from them?**

1.4 Objectives

The main objective of this thesis is **to propose solutions to minimize the execution time of vehicular applications reliably through computation offloading in VEC systems**. This main objective was broken down into specific objectives in order to achieve it. Following, we list such specific objectives:

- To design and implement a VEC system based on mobility and network simulators.
- To implement the simultaneous use of WAVE and 5G technologies in client vehicles of the VEC system.
- To design and implement a new context-oriented framework to support and manage all stages of computation offloading processes.
- To design and implement an offloading failure recovery mechanism.
- To implement fully local execution of applications.
- To investigate and implement literature algorithms to decide task assignments.
- To design and implement new algorithms to decide task assignments.
- To analyze, evaluate, and compare the proposed solutions, the fully local execution of applications, and literature algorithms through simulations in different vehicular environments.

1.5 Methodology

In order to answer the research questions and achieve the objectives of this thesis, the methodological process adopted consisted of the steps described below.

- **Step 1:** Identification of the problem and existing solutions. This step was carried out through extensive bibliographic research. In addition, the mathematical formulation of the problem and the VEC system was also carried out.
- **Step 2:** Preparation of the assessment environment. In this step, we decided to use two simulators widely used in VANETs works: the ns-3 (RILEY; HENDERSON, 2010) as network simulator and the Simulation of Urban Mobility (SUMO) (KRAJZEWICZ, 2010) as mobility simulator. After this decision, we integrated the two simulators and implemented a simulated VEC system using WAVE and 5G technologies.
- **Step 3:** Elaboration and implementation of solution strategies. Here, we designed and implemented the context-oriented framework and its failure recovery mechanism. Then, we designed and implemented three decision and task assignment algorithms for computation offloading in VEC.
- **Step 4:** Evaluation of proposed solutions. In this step, we implemented the fully local execution of applications and literature algorithms for comparison with our solutions. Then, we performed extensive simulations with different vehicular environments to collect the results and evaluate the solutions.

1.6 Contributions

The main contributions of this thesis are:

- **A context-oriented framework for computation offloading in VEC systems.** This framework is responsible for all the support and management of computation offloading processes. It performs resource discovery, gathers contextual information, sends and receives tasks, and provides a failure recovery mechanism. Its main module is the decision maker that allows the coupling of different decision algorithms.
- **Three decision and task assignment algorithms to minimize the execution time of vehicular applications reliably.** These algorithms, listed below, consider several contextual parameters to assess the risks and benefits associated with computation offloading.
  - **Greedy for CPU Free (GCF)**, an algorithm that prioritizes sending tasks to servers with the highest processing availability and the shortest distances to the client vehicle.
  - **Greedy Task by Task (GTT)**, an algorithm that assigns each application task to the best possible server, updating it in real-time after each assignment. To evaluate the servers, it considers the distance to the client and their CPU capacities and

availability.

  - **Artificial Bee Colony (ABC) for Computation Offloading in VEC (BCV)**, a task scheduling and intelligent algorithm based on ABC metaheuristic. This algorithm is inspired by the behavior of honey bees searching for food sources, which represent feasible solutions for the minimization problem in question.

- **Use of a special contextual information about known routes of vehicles**, helping to predict vehicle positioning more accurately and avoid offloading failures.
- **Simultaneous use of WAVE and 5G technologies**, combining their advantages, increasing capacities, and decreasing task transmission delays.

1.7 Publications

This section presents the list of publications related to this thesis as follows.

- **Title:** A Context-Oriented Framework for Computation Offloading in Vehicular Edge Computing using WAVE and 5G Networks.
  Authors: A. B. Souza, P. A. L. Rego, T. Carneiro, P. H. G. Rocha, and J. N. Souza.
  Journal: Vehicular Communications, in press.
  Classification: 96 % (Scopus highest percentile) and A1 (Qualis).
- **Title:** Computation Offloading for Vehicular Environments: A Survey.
  Authors: A. B. Souza, P. A. L. Rego, T. Carneiro, J. C. Rodrigues, P. P. R. Filho, J. N. Souza, V. Chamola, V. H. C. Albuquerque, and B. Sikdar.
  Journal: IEEE Access, volume 8, 2020.
  Classification: 87 % (Scopus highest percentile) and A2 (Qualis).
- **Title:** A Task Offloading Scheme for WAVE Vehicular Clouds and 5G Mobile Edge Computing.
  Authors: A. B. Souza, P. A. L. Rego, P. H. G. Rocha, T. Carneiro, and J. N. Souza.
  Conference: IEEE Global Communications Conference - IEEE GLOBECOM, 2020, Taipei, Taiwan.
  Classification: A1 (Qualis).
- **Title:** Exploring Computation Offloading in Vehicular Clouds.
  Authors: A. B. Souza, P. A. L. Rego, and J. N. Souza.
  Conference: IEEE International Conference on Cloud Networking - IEEE CloudNet, 2019, Coimbra, Portugal.

Classification: A2 (Qualis).

1.8 Thesis Organization

We describe how the rest of this thesis is organized as follows. Chapter 2 presents the background of the principal concepts, such as VANETs, WAVE, 5G, vehicular edge computing, computation offloading, and the artificial bee colony metaheuristic. Chapter 3 provides a taxonomy of the main related works and descriptions and gaps from those works. In addition, discussions and comparisons between our proposed solutions and literature works are provided. Then, Chapter 4 details the system modeling and problem formulation. Chapter 5 presents the proposed solutions in this thesis: a context-oriented framework to support and manage computation offloading processes and three decision and task assignment algorithms. Chapter 6 describes the details of the experiments, parameters, and metrics used. Furthermore, discussions and evaluations of the proposed solutions compared to fully local executions and literature algorithms are shown. Finally, Chapter 7 presents the main conclusions of this thesis, responses to research questions, and directions for future works.

2 BACKGROUND

This chapter contains the fundamental concepts related to this thesis. First, Section 2.1 briefly describes the vehicular ad hoc networks. Section 2.2 presents two of the main vehicular communication technologies currently used. Descriptions of the vehicular edge computing paradigm and its remote execution environments are provided in Section 2.3. In turn, Section 2.4 details the computation offloading technique. Sections 2.5 and 2.6 explain the concepts of context and the artificial bee colony algorithm. Finally, Section 2.7 shows the concluding remarks of this chapter.

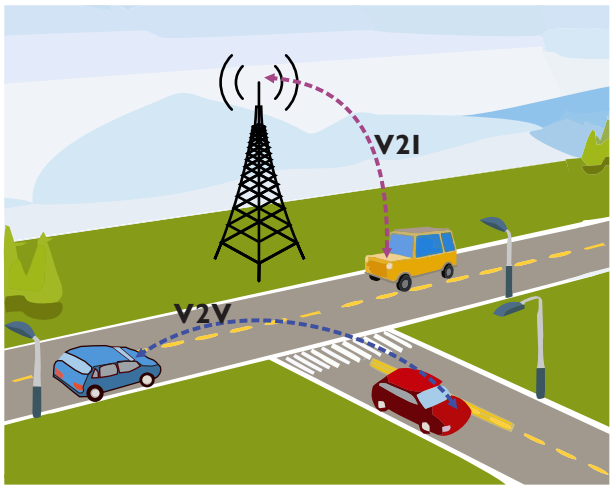
2.1 Vehicular Ad Hoc Networks

Communication systems that enable the transmission and reception of data between different vehicles form the so-called VANETs. In these networks, a vehicle needs an OBU to process and store information and communicate with other devices. Since vehicles have their communication capabilities, these networks do not require a fixed structure and are very useful for Intelligent Transportation Systems (ITS) (SOMMER; DRESSLER, 2014; ALVES *et al.*, 2009).

VANETs have two main types of communication: V2V and V2I (Figure 1). V2V communications allow a VANET to be formed anywhere, requiring only that vehicles be equipped with OBUs and that they are within the communication range of each other. A common type of message in V2V communications is beacon messages, making vehicles aware of their environment by containing information about other vehicles' speed, position, and direction. V2I communications, in general, are used to provide infrastructure access to the Internet. This access is made through RSUs and BSs, which are fixed units located on buildings, shoulders, or sidewalks (SOMMER; DRESSLER, 2014; AL-SULTAN *et al.*, 2014; YOUSEFI *et al.*, 2006).

According to Al-Sultan *et al.* (2014), VANETs have some peculiar characteristics. For example, as the network nodes are the vehicles themselves, there is a certain predictability in their movements. They need to follow the patterns of the traffic lanes, such as speed limits and width of the lanes. Except in combustion-powered vehicles off, there are no significant energy restrictions for network nodes. The transmission can be direct between them, reducing the routing delay and decreasing the overhead at the base stations. Furthermore, as there is no central coordination point or just a communication technology, the systems must be distributed

Figure 1 – Example of a VANET.



Source: The Author.

and interoperable (HARTENSTEIN; LABERTEAUX, 2008).

In addition, the network topology is influenced by the behavior of drivers (such as turning on a street or changing direction) and the wide variation in speed of the network nodes. Thus, nodes can quickly approach or move away from other nodes in the network, generating frequent network disconnections, loss of network packets, and connections with a short duration. Installing more RSUs and BSs can minimize network disconnections and fragmentations. But installing such infrastructures has a high cost and is not a guarantee of total connectivity, as they can be overloaded (AL-SULTAN *et al.*, 2014).

Another consequence of the rapid movement of vehicles is the frequent changes in network topology and sudden changes in the number of nodes in the network. This way, a network can be with few nodes and, in a short time, it can become dense with a large number of connected nodes (AL-SULTAN *et al.*, 2014). This variation in vehicular density can also contribute to a more significant loss of network packets and constitutes a scalability challenge. In general, packet losses happen due to low network connectivity (when there are few vehicles) or packet collisions in the wireless environment (when there are many vehicles) (PANICHPAPIBOON; PATTARA-ATIKOM, 2008; TONGUZ *et al.*, 2007).

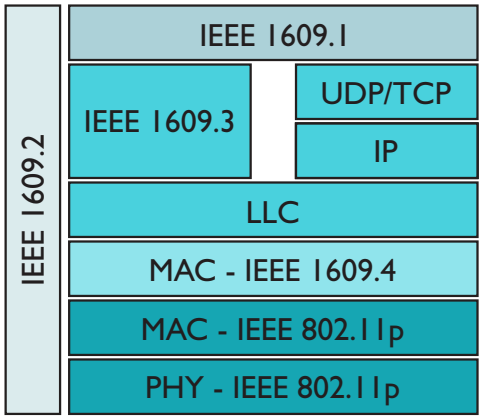
2.2 Vehicular Communication Technologies

Below, we present two of the leading vehicular communication technologies currently used. Section 2.2.1 gives an overview of the WAVE architecture. Then, Section 2.2.2 presents the fifth-generation cellular networks (5G).

2.2.1 WAVE

WAVE is an architecture standardized by the Institute of Electrical and Electronic Engineers (IEEE) for vehicular communications (SOMMER; DRESSLER, 2014; IEEE, 2011; UZCATEGUI; ACOSTA-MARUM, 2009). As shown in Figure 2, the WAVE architecture is a family of protocols defined for different layers of the Internet protocol stack.

Figure 2 – WAVE architecture.



Source: The Author.

Some protocols of the WAVE architecture stand out. For example, IEEE 1609.1 manages the synchronization of OBUs and RSUs and the computational resources of these devices. IEEE 1609.2 defines secure message processing and formats. IEEE 1609.3 is responsible for the network, transport, and Logical Link Control (LLC) layers. Thus, IEEE 1609.3 specifies how to incorporate the Internet Protocol (IP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). Through the LLC, IEEE 1609.3 allows using the TCP/UDP/IP stack or the WAVE Short Message Protocol (WSMP) as an alternative to providing lower latency. IEEE 1609.4, which belongs to the upper part of the Medium Access Control (MAC) layer, enables multi-channel operation and packet prioritization. Finally, IEEE 802.11p defines the physical layer (PHY) and the lower part of the MAC layer. Some of its features are: random MAC address, operating ranges up to 1000 meters, and wildcard Service Set Identifications (SSIDs) (SOMMER; DRESSLER, 2014; UZCATEGUI; ACOSTA-MARUM, 2009; ALVES *et al.*, 2009).

Regarding the band spectrum of the physical layer of the WAVE architecture, a well-accepted standard is called Dedicated Short Range Communications (DSRC) (JIANG; DELGROSSI, 2008). Originally, this standard made exclusive use of frequency bands around 5.9

GHz, allocated in 1999 by the Federal Communications Commission of the United States (FCC) for ITS with V2V or V2I communications. The DSRC spectrum was structured in seven channels of 10 MHz, with one control channel (for WSMP messages of control and security) and the others being available as service channels or for other purposes (UZCATEGUI; ACOSTA-MARUM, 2009).

Current Status of the Technology

Standards related to WAVE architecture have been the dominant vehicular technology and are ready for use after years of testing. This technology is in production in the United States, Europe, and Japan for commercial use. In 2018, there were more than 100,000 vehicles equipped with WAVE standards in Japan and more than 15,000 in the United States in 2020 (ANSARI, 2021; NUTS AND VOLTS MAGAZINE, 2018; FIERCEWIRELESS, 2018; ARS TECHNICA, 2020). Furthermore, the technology continues to be improved. For example, the IEEE formed a working group in January 2019 to develop the IEEE 802.11bd protocol, considered the evolution of IEEE 802.11p (ANSARI, 2021; NAIK *et al.*, 2019).

However, there has been intense dispute over the exclusive use of the 5.9 GHz spectrum for ITS and over the standard technology to be adopted. In the United States, in November 2020, of the original 75 MHz, the FCC reserved only 30 MHz for ITS and designated the transition from DSRC to Cellular Vehicle to Everything (C-V2X) (FCC, 2020; FIERCEWIRELESS, 2020; BLOOMBERG LAW, 2019). In this sense, there have been studies on interoperability or dual-use between DSRC and C-V2X (ANSARI, 2021). In Europe, on the other hand, the trend is the maintenance or expansion of the spectrum for ITS. Regarding the technology adopted, some European countries remain neutral. Nevertheless, the European Telecommunications Standards Institute (ETSI) wants to clarify in mid-2022 how the IEEE 802.11p (already adopted by Volkswagen, for example) and C-V2X can coexist and regulation can occur until September 2023 (HEISE MEDIEN, 2020; INTERNATIONAL RAILWAY JOURNAL, 2020; EUROPEAN COMMISSION, 2020).

2.2.2 5G

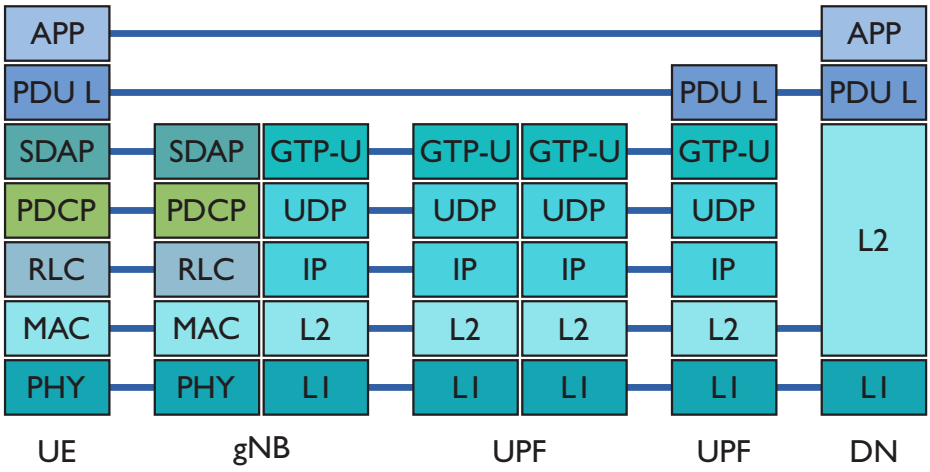
The fifth-generation cellular mobile networks (5G) are wireless communication networks that have high speed and capacity and radio coverage areas divided into cells. Each cell is associated with a base station, which provides connectivity between the Data Network

(DN) and various devices within the cell, including User Equipments (UEs) (NGO *et al.*, 2013).

5G networks are standardized by the 3rd Generation Partnership Project (3GPP) organization and by the radiocommunication sector of the International Telecommunication Union (ITU). These networks were designed to support improved mobile broadband (with the potential to reach up to 20 Gbps of data rate), greater node mobility, and low latency and ultra-reliable communications. For this, 5G networks use beamforming, massive Multiple Input Multiple Output (MIMO) channels, and direct communications Device to Device (D2D) (SHAFI *et al.*, 2017; STORCK; DUARTE-FIGUEIREDO, 2019).

The architecture of these networks is modularized and based on the separation of network functions between functions of the control plane (control access to the radio and the connection of the UE with the DN) and functions of the user plane (transfer user data) (MARSCH *et al.*, 2018). Thus, after the 5G control plane establishes access, a Protocol Data Unit (PDU) session provides user plane connectivity between applications (APPs) of a UE and the DN, as shown in Figure 3.

Figure 3 – Layers and protocols of the 5G user plane.



Source: The Author.

According to Figure 3, for UE data to arrive at the DN, they must first pass through Gigabit NodeB (gNB) and the User Plane Function (UPF). The gNB is the base station of the 5G and is the direct contact with the UE. The UPF module is responsible for packet routing, QoS, application of policies, traffic reports, anchoring radio access mobility. The PDU layer (PDU L) handles information related to the PDU session between the UE and the DN. The PHY and L1 layers refer to the 5G physical layer and the MAC and L2 layers refer to the 5G link layer. The Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layers are

responsible for some MAC layer functions: error correction/detection, segmentation/reassembly, and data integrity protection. The Service Data Adaptation Protocol (SDAP) layer serves as an interface between the core network and the Radio Access Network (RAN), mainly made up of the gNB. Finally, the General Packet Radio Service Tunnelling Protocol for User Plane (GTP-U) layer supports traffic multiplexing from different PDU sessions (PENTTINEN, 2019; CHANDRAMOULI *et al.*, 2019).

Regarding the band spectrum of the 5G physical layer, two frequency bands are considered: below 6 GHz and above 24/30 GHz. 5G networks with frequencies below 6 GHz (called sub-6 GHz) allow long-distance propagation and low penetration loss. However, the sub-6 GHz spectrum has become heavily congested as many technologies use the same spectrum (KRATSIOS, 2019; MEZZAVILLA *et al.*, 2018).

Higher frequency 5G networks (between 30 and 300 GHz) are related to wavelengths between 1 and 10 mm. For this reason, they are called Millimeter Wave (mmWave) networks. These networks have the potential to achieve massive data throughput. Nevertheless, they have a high loss of propagation, penetration, and signal attenuation rate. Thus, 5G networks with this spectrum need visibility close to the line-of-sight. For networks to adapt to these characteristics, they use more antennas (which are also small) to support cells with lower coverage, Multi-User MIMO (MU-MIMO), and beamforming, which improve the antenna gain, range, and efficiency (MEZZAVILLA *et al.*, 2018).

Current Status of the Technology

As mentioned by the last Ericsson Mobility Report (JONSSON *et al.*, 2020), there are already more than 100 providers offering 5G services and more than 150 5G device models available, including devices that support frequency ranges of the sub-6 GHz and mmWave spectrum. It is estimated that around 15 % of the world population already has 5G coverage and it is expected that this number increases to 60 % by the end of 2026. In this regard, the United States, China, South Korea, and Switzerland stand out. The latter managed to get 5G technology to cover more than 90 % of the population in 2019.

Besides, vehicle manufacturers have already launched vehicles with support for 5G technology, allowing V2I or V2V. Globally, the launch of vehicles with 5G capacity faces challenges such as the still low 5G coverage and the dispute with WAVE/IEEE 802.11p technology. Therefore, some manufacturers are designing hybrid solutions, such as the BMW iX

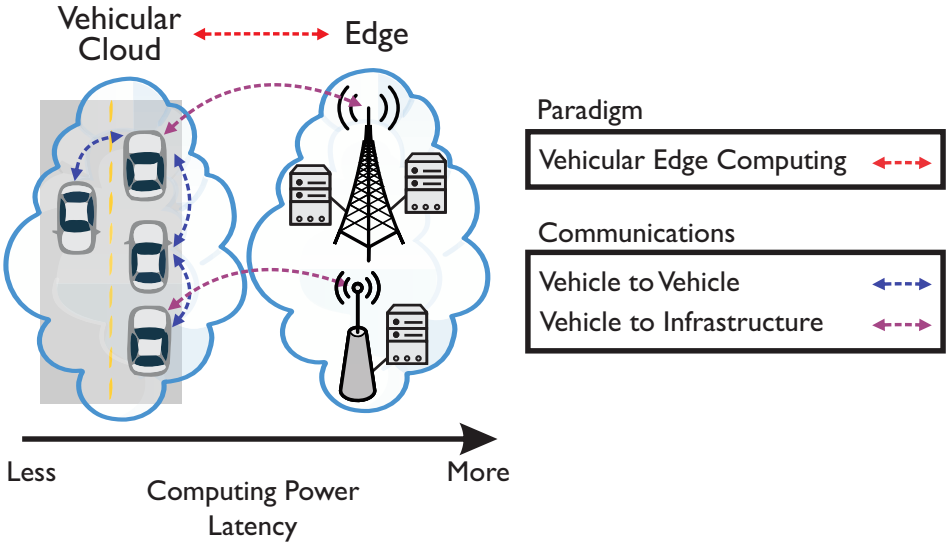
vehicle, which supports cellular communication and WAVE. However, it is estimated that around 5 million vehicles worldwide will support 5G in 2023 (IHS MARKIT, 2020).

About 5G vehicular in the sub-6 GHz frequencies, in the United States and Europe, the spectrum can be used for ITS (AUTOMOTIVE ASSOCIATION 5GAA, 2020; FCC, 2020). Regarding the higher frequency spectrum (mmWave), the United States has made available bands of 24, 28, 37, 39, and 47 GHz for 5G use. In Europe, the spectrum of 24.25-27.50 GHz and 40.5-43.5 GHz have been considered for the 5G use (ORGANIZATION 5G AMERICAS, 2020).

2.3 Vehicular Edge Computing

In accordance with Souza *et al.* (2020), VEC refers to the paradigm that allows two types of remote execution environments to be used in an isolated or integrated way: VC and Edge, as shown in Figure 4.

Figure 4 – VEC paradigm and the integration between a vehicular cloud and the edge.



Source: The Author.

A VC is a set of vehicle computational resources dynamically coordinated to offer services on-demand through V2V connections. Thus, vehicle owners can rent or lend onboard computer resources to other vehicles or clients. Based on the computational availability of the vehicles, these mobile clouds can be integrated with edge devices and remote clouds, or they can be isolated, autonomous, and ad hoc (ABDELHAMID *et al.*, 2017; BOUKERCHE; ROBSON, 2018; WHAIDUZZAMAN *et al.*, 2014; AHMED *et al.*, 2019)

In the context of vehicular networks, the term Edge is used to refer to a set of edge

servers deployed in the vicinity of roads, streets, and avenues by service providers. Edge servers are computational devices coupled to base stations / RSUs one or a few hops away from vehicles. They can operate with little or no Internet connectivity, be isolated or belong to small data centers, and are only available to users within the RAN (SOUZA *et al.*, 2020; ZHOU *et al.*, 2018; RAZA *et al.*, 2019; YOUSEFPOUR *et al.*, 2019; PHAM *et al.*, 2020).

Thus, in VEC, computational processing can be distributed between VCs and edge servers. With more processing options, systems can take full advantage of both remote execution environments (ZHOU *et al.*, 2018; RAZA *et al.*, 2019). For example, using a VC, a VEC system can achieve low-cost computing, distributed processing in non-infrastructure scenarios, real-time services, and reduced latency through direct connectivity. Using edge servers, the VEC system achieves greater processing power and storage capacity, energy savings for vehicles, reduced packet traffic between the core and the periphery of the network, and improved QoS of vehicular applications through faster processing and few network hops (SOUZA *et al.*, 2020).

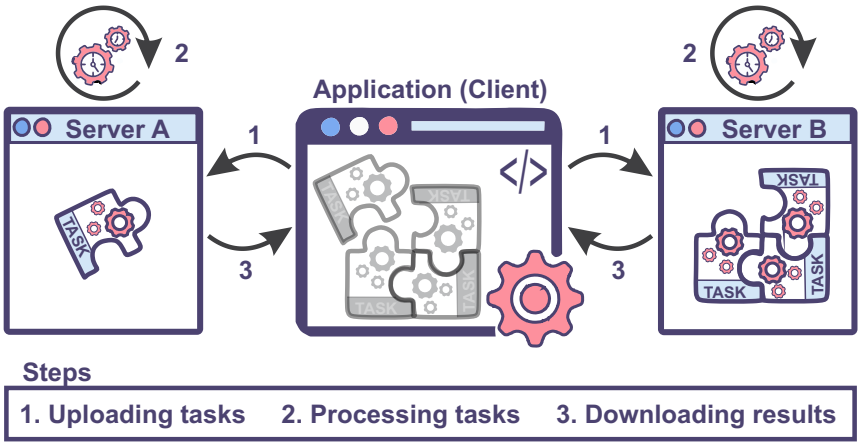
Nonetheless, a VEC system suffers from the dynamism and mobility of the vehicular environment. Also, its computational resources are not as powerful as those of the traditional cloud. In fact, the central cloud permanently stores massive data, performs complicated computing tasks, and has theoretically unlimited computational resources. Even so, vehicular applications that are sensitive to delay may not tolerate the highest latency to access this type of remote cloud and may not always have access to the Internet. Therefore, these applications are best executed on VEC systems (SOUZA *et al.*, 2020; KANG *et al.*, 2018).

2.4 Computation Offloading

Computation offloading is a technique that consists of sending tasks or parts of an application to remote servers or devices so that they are executed and the results returned to the original application. These servers have available and accessible computing resources (REGO *et al.*, 2017; XU *et al.*, 2018). Figure 5 shows this process. In step 1, the application offloads computation tasks to two remote servers. In step 2, the servers process the tasks. Ultimately, in step 3, the client application downloads the processing results.

When migrating parts of an application or the complete application, the main objective of this technique is to improve the execution time of applications. However, other benefits can also be obtained. For example, the computational offloading process can save device energy and decrease processing overheads by transferring the execution of heavy workloads to remote

Figure 5 – Figurative depiction of the computation offloading technique.



Source: The Author.

servers (SOUZA *et al.*, 2020; HASSIJA *et al.*, 2020; YOUSAFZAI *et al.*, 2019).

Some steps are necessary for the computation offloading to be carried out, such as resource discovery, application partitioning, and decision making (SHARIFI *et al.*, 2012). Discovery may be automated, or the computing resources may have been informed in advance (REGO, 2016). In partitioning, the application is divided into tasks that can be executed on different devices. This partitioning can be done through different models, techniques, and levels of granularity. Also, partitioning can be done automatically by the system or manually by the application developers through annotations or markup in the source code (SOUZA *et al.*, 2020).

The decision-making step must define where, when, and what the application's tasks should be sent. This is the most critical and challenging step in the computation offloading process. If the application developer makes these definitions before the application is run, the process is classified as static. If the system makes the definitions at run time, the process is classified as dynamic (KUMAR *et al.*, 2013). Dynamic decisions aiming to distribute tasks optimally require considerable computational resources and are described as NP-hard problems (SOUZA *et al.*, 2020). These decisions are usually based on several metrics. Some metrics about the current condition of the network used are latency, communication range, connection type, and packet loss. Application metrics used are the size of data to be transferred, execution time, and component dependency. Finally, metrics about network nodes commonly used are memory, CPU, battery, distance, and speed. These metrics can be collected at run time or through historical data.

When deciding where to process tasks remotely, the computation offloading system can consider the following options: traditional cloud, edge servers, and vehicular cloud (SOUZA

et al., 2020). The advantages and disadvantages of these remote execution environments are discussed in Section 2.3. Though, the computation offloading technique is not always worthwhile. For example, when the local execution time estimate is less than the time to send tasks, process remotely, and receive the results, offloading is not advantageous (KUMAR *et al.*, 2013).

2.4.1 Computation Offloading vs. Data Offloading

Although there are similarities between the techniques of computation offloading and data offloading, they should not be confused. In data offloading, the main objective is to replicate popular data in locations closer to users, in an approach similar to that of Content Delivery Networks (CDNs). In this sense, some mobile network nodes download popular content and transmit the content to other nodes, reducing redundant traffic on congested networks, network delays, energy consumption, and financial costs (XU *et al.*, 2018).

Notwithstanding, the computation offloading process consists of more steps, which are: uploading what needs to be processed, processing itself, and downloading the computation result. On the other hand, the data offloading consists mainly of only one step: downloading the content without computation and return of results. Besides, in computation offloading, the content (computation result) popularity is typically zero, serving only for one user/device. In data offloading, the popularity of content is typically high, with many users requesting downloads. Finally, while in data offloading, the contents are relatively large (e.g., audios, images, and videos), the contents are relatively small in computation offloading (CHEN *et al.*, 2016).

2.5 Context

Context refers to information that characterizes entities, including the status of network devices or vehicles. Contextual information assists decision-making processes and improves the performance of systems, allowing adaptations to different circumstances and environments (XU *et al.*, 2018).

According to Perera *et al.* (2013), this contextual information can be divided into two types: primary and secondary context. Primary context is any information acquired directly without any data processing or fusion. This type of contextual information is also called low-level context and is directly related to raw data. An example of a primary context would be satellite-based positioning information. The secondary context, or high-level context, is any

information derived from the primary context through data processing or fusion. For example, the distance between two vehicles is considered secondary context information because it is obtained through calculations that use the location parameters of two vehicles (primary context) (PERERA *et al.*, 2013; YÜRÜR *et al.*, 2014). Also, context data can be classified as static or dynamic. Static contextual data does not change over time. Examples of such data are unique identifiers and the computing capabilities of devices. Dynamic contextual information changes over time (PERERA *et al.*, 2013). For example, location, speed, and processing queue time are dynamic contextual data.

Systems that use contextual information to adapt their operations according to situations or environments are called context-aware or context-oriented. The life cycle of this type of system generally follows the following steps: context acquisition, processing, and acting. Context acquisition gathers primary context information and raw data directly from sensors or remote devices (RAZA *et al.*, 2019). Context acquisition can be pull-based, where the interested system requests contextual data, or push-based, where sensors automatically send context data to the interested system. This acquisition can also be made eventually or periodically. In the eventual form, the acquisition of context is only carried out through a trigger actioned by some event or change of circumstance. In periodic form, context data is acquired regularly at a specific frequency (PERERA *et al.*, 2013).

The processing step, in context-oriented systems, consists of applying reasoning or inferencing techniques to obtain secondary or high-level contextual information. Context reasoning or inferencing techniques deduce new knowledge and better understanding based on previous contextual information. In this step, the raw data is improved in the pre-processing phase. The system then combines or merges diverse context data from multiple devices to produce more accurate and complete information. In the last phase of this step, context inference is performed, generating high-level context information from lower-level context. Finally, acting, the last step of a context-oriented system's life cycle, involves applying context inference for adaptations, data distributions, or decision making (PERERA *et al.*, 2013; RAZA *et al.*, 2019).

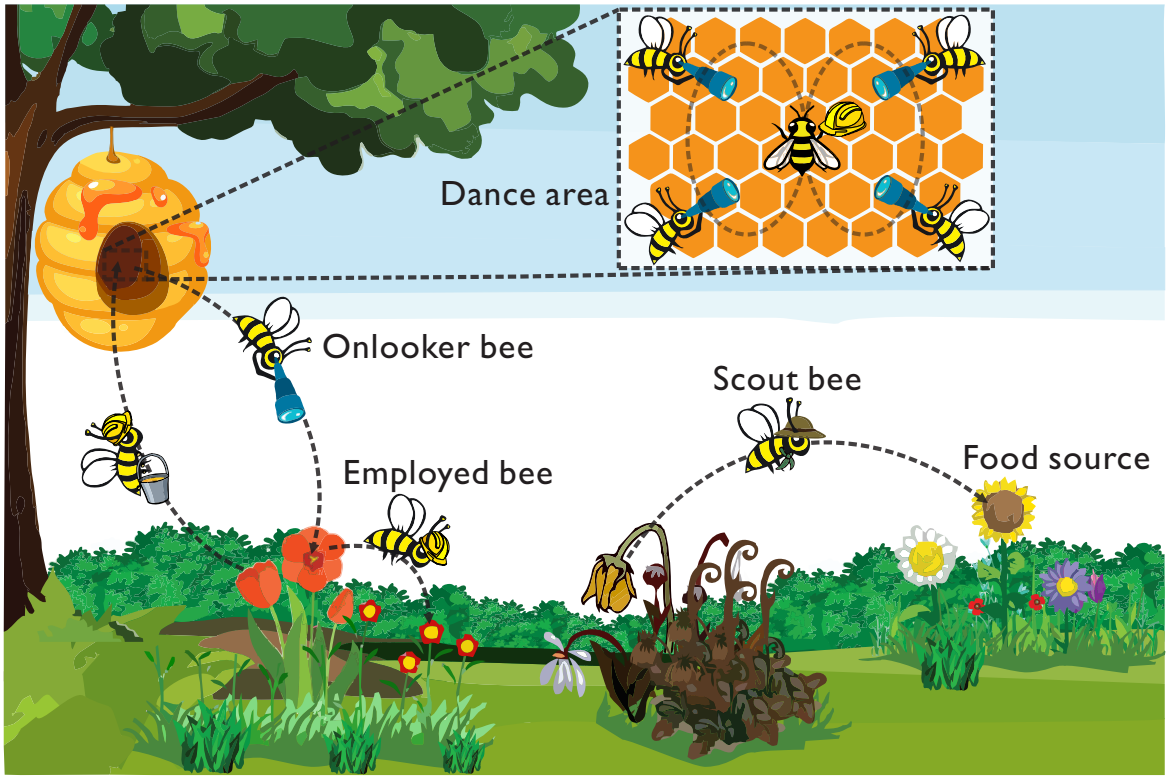
2.6 Artificial Bee Colony

Artificial Bee Colony (ABC) is a metaheuristic optimization algorithm inspired by the foraging behavior of swarms of honey bees. ABC was proposed by Karaboga (2005) and is used to solve multi-objective and multi-variable optimization problems (KARABOGA; AKAY,

2009; KHOSRAVANIAN *et al.*, 2018).

The ABC algorithm is modeled according to the behavior of three types of bees in search of food sources: employed, onlooker, and scout bees, as shown in Figure 6. An employed bee is a bee that is exploring a food source and its neighborhood. This type of bee shares information about food sources with other bees. This sharing takes place through a dance (called waggle dance), which indicates the location and profitability of the food source. The longer the dance lasts, the greater the amount of nectar of the food source (KARABOGA; AKAY, 2009; HE *et al.*, 2014; GAO *et al.*, 2016).

Figure 6 – Figurative depiction of a artificial bee colony.



Source: The Author.

The bees that are in the hive watching the dances of the employed bees are called onlooker bees. They decide to explore certain food sources based on the observations of these dances. The better the food source, the more likely an onlooker bee will choose it to exploit it. When a food source runs out, an employed bee leaves it and becomes a scout bee. So, this scout bee does a random search for new food sources (KARABOGA; AKAY, 2009; ZHANG *et al.*, 2014; CHEN; XIAO, 2014).

In the ABC algorithm, the position of a food source represents a possible solution to a given optimization problem. The amount of nectar from the food source corresponds to the

quality or fitness of the solution. Furthermore, the ABC algorithm has four main phases: (1) initialization, (2) employed bees, (3) onlookers bees, and (4) scout bees (HE *et al.*, 2014). After the initialization phase, phases 2 to 4 are repeated until a maximum number of cycles is reached, or a stopping criterion is met (KARABOGA; AKAY, 2009). In the following, we describe the four phases.

Phase 1: Initialization. Scout bees do a random global search to find an initial set of solutions. Then, the solutions are evaluated and receive a fitness value (KHOSRAVANIAN *et al.*, 2018; HE *et al.*, 2014). After finding the initial set of solutions, these scout bees become employed bees, each one is associated with a solution, and the algorithm starts phase 2.

Phase 2: Employed Bees. Employed bees search for other solutions in the vicinity of their associated solutions, making variations on them. After finding a new solution, each employed bee evaluates its quality through a fitness value. Suppose the new solution has better fitness than the previous one. In that case, the employed bee abandons the previous solution and memorizes only the new solution. Otherwise, the previous solution remains in memory (ZHANG *et al.*, 2014). Upon returning to the hive, the employed bees share information about the solutions and their fitness for the onlooker bees in the dance area (KARABOGA; AKAY, 2009).

Phase 3: Onlooker Bees. The onlooker bees observe the information about solutions provided by the employed bees. After observation, they decide which solution to explore using a fitness-based probabilistic selection (e.g., roulette wheel, tournament selection, or ranking based). Better fitness solutions are more likely to be chosen. After associating themselves with the chosen solutions, onlooker bees follow the same procedure as employed bees, searching for new solutions in the vicinity of current solutions (ZHANG *et al.*, 2014; CHEN; XIAO, 2014).

Phase 4: Scout Bees. When an employed bee realizes that its search space is not generating better fitness solutions for a certain number of cycles, it abandons that solution and its search space. Then, it becomes a scout bee and starts searching for solutions in new search spaces randomly (KHOSRAVANIAN *et al.*, 2018).

Therefore, in phases 2 and 3, the ABC algorithm uses local search in a given region of the set of solutions, thus making exploitation (or intensification of searches in a region). In phases 1 and 4, it uses global search to find new solution spaces, thus exploring, diversifying, and helping to avoid suboptimal solutions and premature convergences (KARABOGA; AKAY, 2009; CHEN; XIAO, 2014).

The ABC algorithm has advantages in being applied to optimization problems. For example, compared to other conventional metaheuristic algorithms, ABC employs fewer control parameters and can quickly return good solutions to problems that need to be solved in real-time (KARABOGA; AKAY, 2009). It can also solve NP-hard optimization problems, returning solutions close to the global optimum (KHOSRAVANIAN *et al.*, 2018). Besides, ABC is simple, easy to implement, and has shown better performance than other intelligent algorithms concerning the convergence time and the ability to avoid optimal local solutions (CHEN; XIAO, 2014; ZHANG *et al.*, 2014).

2.7 Concluding Remarks

This chapter presented the main concepts related to this thesis. For example, we saw the definition of VANETs, their types of communications, their peculiar characteristics, and their challenges. We also examined the architectures, protocols, and frequency spectrum of WAVE and 5G technologies, as well as their current status. These technologies enable the formation of VANETs. Then, this chapter described the VEC paradigm, which allows computation tasks to be distributed to vehicular clouds and edge servers through VANETs. Systems that use VEC have different computational resources at their disposal and can avoid the excessive latency of access to traditional cloud servers.

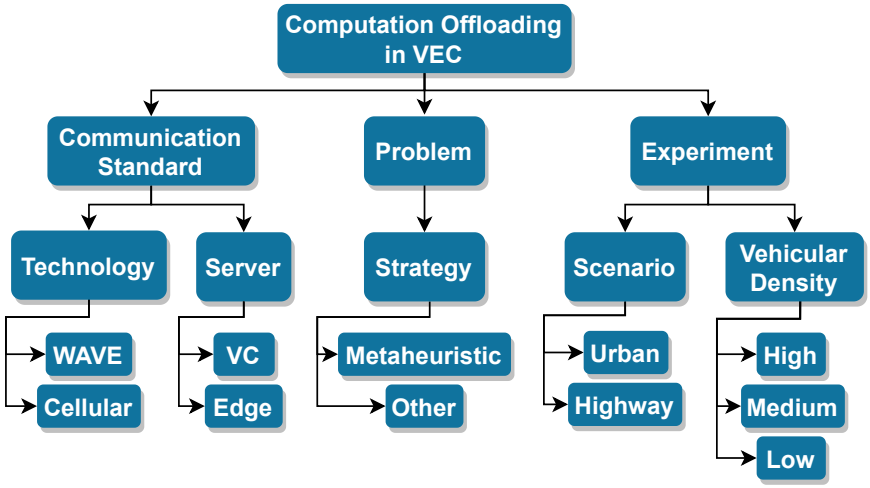
After, the concept and details of the computation offloading technique were presented. Moreover, the difference between computation offloading and data offloading was explained. As we will see in the following chapters, the combination of computation offloading and VEC makes it possible for vehicular applications to reduce their execution times and the processing overhead of VANET nodes. Subsequently, we saw definitions of context and context-oriented systems, context types, and context life cycle issues. In our proposal, different context parameters are used to deliver better results in computation offloading processes. Finally, we described the ABC metaheuristic, presenting its biological inspiration, details and the different phases of the algorithm, and the advantages of its application in optimization problems. An adapted version of this algorithm for computation offloading in VEC is defined in Chapter 5.

3 LITERATURE REVIEW

Several works have been developed in the area of offloading in vehicular networks. For example, in Souza *et al.* (2020), we have done an extensive review and classification of several solutions of computation offloading in VANETs. Among these solutions, some consider that the clients (devices with applications that need computation offloading) are edge servers (SUN *et al.*, 2018; BOUKERCHE; SOTO, 2020; LI *et al.*, 2020a) or pedestrian smartphones (WANG *et al.*, 2018; PHAM *et al.*, 2019; ZHOU *et al.*, 2019). Some solutions do not consider the VEC paradigm (allowing the use of traditional cloud infrastructure) (ZHANG *et al.*, 2019; WU *et al.*, 2019; DU *et al.*, 2018), and others consider other objectives such as reducing energy consumption (MIDYA *et al.*, 2018; GU; ZHANG, 2021; GU; ZHOU, 2019) and financial cost (MISRA; BERA, 2019; TAN *et al.*, 2019; WANG *et al.*, 2020).

Although there are different aspects in this field of research, this chapter specifically presents the principal works of computation offloading in VEC that have two essential characteristics. The first is that clients are only vehicles. The second is that the proposed solutions have a single objective: to reduce vehicular applications' execution time. The core of this chapter is based on a taxonomy that we propose to classify different research articles in this area. The main parts of this taxonomy, shown in Figure 7, are: Communication Standard, Problem, and Experiment.

Figure 7 – Proposed taxonomy of computation offloading in VEC.



Source: The Author.

The nodes of this taxonomy tree guide the reading of this chapter. They indicate the sections where each category is discussed. Thus, Section 3.1 presents the research articles

from the perspective of communication standards, categorizing the works concerning the types of communication technologies used and servers where the computational tasks are executed. Section 3.2 shows the works according to the research problem approached and the main strategies used to solve it. Descriptions of the experiments carried out to validate the works are shown in Section 3.3. The experiments, consisting of simulations, are classified under two aspects: type of scenario and vehicular density. Each of these sections contains a subsection that discusses the gaps of existing works and compares them with the solutions proposed in this thesis. Finally, Section 3.4 presents the final considerations of the chapter.

3.1 Communication Standard

In offloading processes, choosing how to communicate is an important decision. This section analyzes previous work of computation offloading in VEC concerning two aspects of the communication standards: technology and type of server used. After, we discuss the works related to our proposal.

3.1.1 Technology

Different technologies have been proposed to enable device communication and provide reliable and efficient computation offloading in vehicular environments. Below, we report the leading technologies that the works of the area have used.

WAVE

The most used communication technology for works of computation offloading in VANETs is the WAVE (SOUZA *et al.*, 2020). In some of these works, WAVE is the exclusive technology used to transfer computation offloading files. For example, Sun *et al.* (2019b) proposes the IEEE 802.11p and IEEE 1609.4 protocols of the WAVE architecture to form the vehicular clouds. In Feng *et al.* (2017), the authors also used the WAVE IEEE 802.11p protocol exclusively for connections between vehicles participating in offloading computation processes. The authors argue that this communication technology enables fast link configurations without having to create a basic service set, reducing overhead when sending data to other vehicles. Chen *et al.* (2020) used the IEEE 802.11p/WAVE standard to formulate the communication model. In this model, DSRC channels allow the propagation of advertisement messages, information about

tasks that need to be processed remotely, and communication between vehicles in the vehicular cloud.

Cellular

Cellular networks are the second most used communication technology in computation offloading in VANETs. Although third-generation cellular mobile networks (3G) have been used in some works, we focus on the most used networks in the offloading area: the fourth-generation cellular mobile networks (4G) and fifth-generation cellular mobile networks (5G) (SOUZA *et al.*, 2020). In this sense, some works use 4G or 5G cellular networks as the exclusive communication technology for information exchange of computation offloading. For instance, the proposal of Rahman *et al.* (2020) uses only mmWave transmissions associated with high frequency 5G networks to model the communication of offloading processes. In Sun *et al.* (2019a), only 4G technology was used to transmit computation offloading data between vehicles. In this case, the spectrum was divided into resource blocks in the time and frequency domain for data sharing between VANETs vehicles.

Hybrid

Other papers used more than one technology to perform computation offloading in VEC systems (NING *et al.*, 2018; QIAO *et al.*, 2018). In Wang *et al.* (2018), the authors used the IEEE 802.11p protocol of the WAVE architecture to exchange offloading data in V2V communications. For offloading processes involving V2I communications, they used 4G technology. The computation offloading work of Feng *et al.* (2018) allows the use of multiple communication technologies. In this case, WAVE and 5G technologies can be used to transfer offloading data between vehicles and between vehicles and RSUs. Offloading to base stations is only allowed through 4G networks.

3.1.2 Server

In VEC systems, two types of servers can execute tasks of computation offloading. The first types are vehicles in the context of VCs. The second types are edge servers coupled to RSUs or base stations. Following, we describe how offloading articles have used these types of servers.

VC

Servers in VCs are the vehicles themselves. They can request remote processing, or they can process computational tasks of other devices. Some works have exclusively used VC vehicles as computation offloading servers (RAHMAN *et al.*, 2020; SUN *et al.*, 2019a; FENG *et al.*, 2017). For example, Sun *et al.* (2019b) only consider servers in VCs to execute computational tasks for other vehicles. These server vehicles are called service vehicles (SeVs) and have enough computational resources to help process tasks. In Hou *et al.* (2016), the authors use vehicles, parked or moving, as infrastructure for communication and computing. In this way, VCs can be created, and offloading tasks can be cooperatively distributed among participating vehicles to be processed. Nabi *et al.* (2017) propose creating VCs with computational resources of multiple vehicles to process tasks. These tasks can be scheduled to different vehicles in a VC.

Edge

This type of server is located at the edge of the network, directly connected to base stations and RSUs. The edge can contain one or more servers or even mini data centers to assist computation offloading clients. Some papers only used edge servers to execute computation offloading tasks. In Ning *et al.* (2018), vehicles can send their computation tasks only via RSU or base station, so that edge servers process them. In the computation model of Liu *et al.* (2020), only edge servers coupled to RSUs process offloading tasks from client vehicles. These computing tasks can be sent to different edge servers. Each RSU is equipped with several edge servers. Zhang *et al.* (2020) propose that client vehicles with computing tasks can offload them to execute only on edge servers embedded in base stations. After processing the tasks, the edge server sends the result back to the client through its original base station or a base station closer to the client.

Hybrid

The hybrid approach happens when more than one server type, previously mentioned, acts as offloading servers for offloading computation processes (WANG *et al.*, 2018; FENG *et al.*, 2018; LI *et al.*, 2014). Vehicles in VCs and edge servers work collaboratively to share resources and process computational tasks of client vehicles in Qiao *et al.* (2018). In this proposal, RSUs and BSs provide powerful computational resources through their respective edge servers. These

servers are abstracted as the pooling resources to improve their utilization. In Lin *et al.* (2017), the computational resources of vehicles and edge servers connected to RSUs are combined to provide a resource pool. Through this resource pool, computational tasks can be executed on edge servers and vehicles in VCs.

3.1.3 Discussion

Table 1 summarizes the communication standard aspects of the works mentioned in this chapter to facilitate the discussion.

Table 1 – Summary of communication standard aspects of related works.

Reference	**Technology**			**Server**	
	WAVE	**Cellular**	**Simultaneous Use**	**VC**	**Edge**
(LI *et al.*, 2014)	√			√	√
(HOU *et al.*, 2016)	√			√	
(FENG *et al.*, 2017)	√			√	
(LIN *et al.*, 2017)	√			√	√
(NABI *et al.*, 2017)				√	
(QIAO *et al.*, 2018)	√	4G		√	√
(WANG *et al.*, 2018)	√	4G		√	√
(NING *et al.*, 2018)	√	5G			√
(FENG *et al.*, 2018)	√	4G/5G		√	√
(SUN *et al.*, 2019b)	√			√	
(SUN *et al.*, 2019a)		4G		√	
(RAHMAN *et al.*, 2020)		5G		√	
(CHEN *et al.*, 2020)	√			√	
(LIU *et al.*, 2020)	√				√
(ZHANG *et al.*, 2020)		√			√
GCF	√	5G	√	√	√
GTT	√	5G	√	√	√
BCV	√	5G	√	√	√

Source: The Author.

As shown in Table 1, computation offloading servers can be located on a VC or the Edge. These servers can receive tasks by different communication technologies such as WAVE, 4G, or 5G. In Rahman *et al.* (2020), authors present solutions to allow vehicles to execute computation tasks on selected neighboring vehicles over only 5G/V2V connections. Other works used similar communication standards but with approaches only either WAVE/V2V (CHEN *et al.*, 2020; SUN *et al.*, 2019b; FENG *et al.*, 2017), or 4G/V2V (SUN *et al.*, 2019a), or WAVE/V2I (LIU *et al.*, 2020). Nevertheless, using only one communication technology limits the capabilities to transmit more network packets and prevents connections to other types of

networks (CHARITOS; KALIVAS, 2017; DREYER *et al.*, 2016; UCAR *et al.*, 2015). Another interesting work in Ning *et al.* (2018) proposes a data and computation offloading scheme for WAVE and 5G networks. This scheme allows tasks to be executed only on edge servers coupled to base stations or RSUs. However, these proposed solutions allow third-party tasks to be executed only by vehicles or only by edge servers. In either case, there is a waste of computational resources from unutilized servers.

Some works allow third-party tasks to be executed by vehicles and edge servers to avoid such a waste of resources. Besides, to increase the network's transmission capacities, these works use WAVE and 4G technologies. In Qiao *et al.* (2018) and Wang *et al.* (2018), the authors propose schemes and algorithms that enable a vehicle to transfer its computations tasks to neighboring vehicles (via WAVE/V2V) or nearby edge servers (via 4G/V2I). Then, the latter can process the tasks and return their results to the requesting vehicle. Nonetheless, while it is an interesting approach, 4G networks are already starting to become outdated. They have deficiencies in relation to 5G networks, such as lower data rates, weaker for high mobility, fewer frequency ranges, and higher latencies. Also, algorithms and techniques used in 4G networks may not be well adapted to 5G networks because of differences in network architecture, modulation and multiplexing techniques, wireless signal behavior, coverage area, and need for a line of sight between devices (BARB; OTESTEANU, 2020; RAPPAPORT *et al.*, 2017; TEHRANI-MOAYYED *et al.*, 2020; MEZZAVILLA *et al.*, 2018). The solution proposed by Wang *et al.* (2018) also does not allow simultaneous WAVE/4G transmissions, wasting time in the execution of tasks.

Feng *et al.* (2018) also considers sequential transmissions. The authors propose a framework to take advantage of several computational resources available locally, in nearby vehicles (via V2V), and on edge servers (via V2I). WAVE, 4G, and 5G technologies can be used. Even so, there is a prioritization of sending tasks via 5G, spending more time than if it were possible to send tasks simultaneously through multiple technologies. Only if 5G fails, the algorithm in Feng *et al.* (2018) will need to seek resources via WAVE, which ends up generating more delays in the offloading process. Moreover, in scenarios where only base stations and few vehicles use 5G, the sending of tasks will end up being, mostly, only via V2I and to the edge servers. Thus, this prioritization by 5G can overload the edge servers and underutilize the vehicles' computational resources. Besides, sending via 5G can result in failures due to inherent technology challenges, such as limited range and line-of-sight issues between devices.

In a different way, our proposed framework and algorithms aim to fill the gaps left by the mentioned works. In terms of technology and servers, we allow tasks to be sent to edge servers and vehicles simultaneously through 5G/V2I and WAVE/V2V connections. With this, we increase the network's transmission capacities and take advantage of all available computing resources.

3.2 Problem

The articles on computation offloading in VEC propose different formulations of problems. Among these problems, the most studied are those that deal with the objective of reducing the execution time of vehicular applications (SOUZA *et al.*, 2020). Although some of these applications have become popular, they are computationally complex, intensive, and real-time. Taking too long to process an application's tasks can compromise its performance, data validity, and even the safety of humans in a vehicle. Thus, reducing the execution time of applications is the main objective of the computation offloading technique.

Although this technique has several steps, one of the most important and challenging to reduce the execution time of vehicular applications is the task distribution. This step can have several parameters and metrics that need to be taken into account to obtain the best possible performance in offloading. Among these parameters are contextual information about transmission and processing delays (e.g., mainly in already overloaded devices) and vehicular mobility. In fact, finding the optimal way to distribute tasks to have the maximum reduction in application processing time has been described as a NP-hard problem (ZHU *et al.*, 2018; FENG *et al.*, 2017). Therefore, it might be costly to solve more extensive instances of such problems to optimality.

Thus, the literature works propose solution strategies to deal with these challenges that range from simple algorithms to complex mathematical modeling and intricate machine learning and metaheuristic algorithms. Below we present the main strategies of the algorithms used in the computation offloading works in VEC. Then, we compare the related works to the algorithms proposed in this thesis.

3.2.1 Strategy

Following the taxonomy, in this section, we deal with the algorithmic strategies used for solving the problem at hand. Next, we see the most common strategy found in the related works: metaheuristic algorithms. After, we comment about other used strategies.

Metaheuristic

Metaheuristics are algorithms that do not guarantee to deliver a proved optimal solution to a given instance of a problem, but usually return a good solution in a feasible time (TALBI, 2009). More specifically, metaheuristics are general and higher-level algorithms that incorporate operators designed to avoid getting stuck in a local optimum, called intensification and diversification operators (BLUM; ROLI, 2003).

In Feng *et al.* (2017), the authors propose a scheduling algorithm based on Ant Colony Optimization (ACO) to solve the task distribution problem in vehicular environments without infrastructure. ACO is a metaheuristic inspired by the behavior of ants in search of food. They argue that their algorithm can find near-optimal solutions in a reasonable amount of time. The authors of Sun *et al.* (2019a) modeled the problem as a knapsack problem and designed a task scheduling algorithm based on the bat metaheuristic. This metaheuristic is inspired by the behavior of micro-bats that use echolocation to avoid obstacles and hunt prey. Modifications were proposed to improve the algorithm and get a good solution quickly in vehicular scenarios without infrastructure. Chen *et al.* (2020) proposed an algorithm based on the Particle Swarm Optimization (PSO) metaheuristic to find the best task allocation scheme and minimize the task execution time. Such an algorithm is mentioned as fast and accurate, contributing to the reduction in task execution time.

Other

In addition to strategies based on metaheuristic algorithms, other strategies have also been used to solve the problem of reducing the execution time of vehicular applications through computation offloading. For example, Qiao *et al.* (2018) proposes clustering so that leader vehicles maintain control over a vehicular cloud, managing their computing resources. When vehicles need to offload tasks, they need to send them to the VC leader first. Thus, the leader can evaluate proposed task assignments and adjust them for the overall benefit of the

network. In Sun *et al.* (2019b), the authors propose a machine learning-based solution to guide the task offloading and minimize the average offloading delay. In the proposal, the solution allows vehicles to learn about the offloading delay performance of their neighboring vehicles while offloading computation tasks. The solution also allows for a balance between exploring different actions and exploiting the learned information. In other papers, greedy (WANG *et al.*, 2018; FENG *et al.*, 2018) or approximation (NABI *et al.*, 2017) algorithms were designed for different and specific modeling of the problem.

3.2.2 Discussion

Some of the works mentioned in this chapter propose only ideas (HOU *et al.*, 2016) or exact strategies (ZHANG *et al.*, 2020; WANG *et al.*, 2018; LIN *et al.*, 2017) to solve the problem of application execution time reduction through computation offloading in VEC. While guaranteeing the optimal solution of the problem, these exact strategies are impractical when the problem is NP-Hard or large scale, or there are not enough computational resources (ZOBOLAS *et al.*, 2008). Sun *et al.* (2019b) proposes an online learning-based strategy to reduce the processing time of vehicle applications. However, this strategy has some challenging characteristics: possible high processing time, large amount of data, high susceptibility to errors, and possible hard-to-access, distributed, unreliable, or poor quality data. In VANETs, these characteristics are even more challenging because the computation resources of vehicles are limited, latency requirements are critical, and the network configuration is highly dynamic (BARH, 2020; YE *et al.*, 2018; L'HEUREUX *et al.*, 2017).

Nevertheless, most works use heuristic solutions, i.e., algorithms specifically designed to deal with a certain problem and provide approximate near-optimal solutions (GUAN, 2020). Other works use a clustering approach (QIAO *et al.*, 2018; NING *et al.*, 2018). However, this approach has disadvantages as uneven energy consumption, long delays in cluster head failure, and bottleneck point in cluster head if the cluster has many members (DUAN *et al.*, 2016; UCAR *et al.*, 2015; ZHAO *et al.*, 2018). Even though heuristic methods do not use clustering, they are not always the best strategies available. For example, in general, metaheuristics (problem-independent techniques) perform better than simple heuristics (problem-dependent techniques). Furthermore, solutions produced by metaheuristics have better quality and greater robustness than those obtained by heuristics techniques (BLOCHO, 2020; KUNCHE; REDDY, 2016; ZOBOLAS *et al.*, 2008).

Thus, some works propose the use of metaheuristics (FENG *et al.*, 2017; SUN *et al.*, 2019a; CHEN *et al.*, 2020). However, the proposed metaheuristics have drawbacks. For example, the ACO metaheuristic changes the probability distribution with the iterations and has sequences of random decisions, making the analysis of the algorithm's behavior difficult. The Bat algorithm cannot efficiently handle discrete optimization problems and requires tuning of several parameters. The PSO metaheuristic suffers from premature convergence and loss of population diversity. On the other hand, our BCV algorithm is based on the ABC metaheuristic, a robust and flexible algorithm that can quickly return near-optimum solutions. Moreover, this metaheuristic is simple and easy to implement, requires few parameters for its configuration, and has outperformed other intelligent algorithms (EZUGWU *et al.*, 2020; KHOSRAVANIAN *et al.*, 2018; CHEN; XIAO, 2014; ZHANG *et al.*, 2014).

Besides, some computation offloading strategies use **contextual information** that aids decision-making processes and helps in adapting to different environments and circumstances (XU *et al.*, 2018). One such piece of information is the CPU availability that can be advertised to neighboring devices. This information indicates whether the CPU is idle or busy. If it is busy, it is usually advertised the waiting time needed to process a new task (RAHMAN *et al.*, 2020). In this way, a device with good CPU availability is more likely to be chosen as a server. Although most of the related work in this chapter has not used this information, we used it in our three proposed algorithms.

Moreover, most of the works in this chapter do not have failure recovery mechanisms, even knowing that a failure can lead to increased latency, incomplete information, and application crashes. In our proposal, the support framework provides this functionality for any coupled decision algorithm. Thus, the GCF, GTT, and BCV work together with the proposed failure recovery mechanism, providing reliability to computation offloading processes.

Table 2 summarizes the aspects discussed in this section.

3.3 Experiment

It is essential to evaluate new proposals (algorithms, schemes, frameworks, and systems) of computation offloading in different vehicular environments for reliable validation. This section reviews the details of the experiments that previous works have used in their proposals related to two aspects: scenario and vehicular density. Discussions about the gaps of the works mentioned and comparisons with our proposal are made next.

Table 2 – Summary of problem aspects of related works.

Reference	Strategy	Contextual Information	Support
	Metaheuristic	**CPU Availability**	**Failure Recovery**
(LI *et al.*, 2014)			
(HOU *et al.*, 2016)			
(FENG *et al.*, 2017)	ACO	√	
(LIN *et al.*, 2017)		√	
(NABI *et al.*, 2017)			
(QIAO *et al.*, 2018)			
(WANG *et al.*, 2018)			
(NING *et al.*, 2018)			
(FENG *et al.*, 2018)		√	
(SUN *et al.*, 2019b)			
(SUN *et al.*, 2019a)	Bat		
(RAHMAN *et al.*, 2020)		√	
(CHEN *et al.*, 2020)	PSO		
(LIU *et al.*, 2020)		√	
(ZHANG *et al.*, 2020)			
GCF		√	√
GTT		√	√
BCV	ABC	√	√

Source: The Author.

3.3.1 Scenario

The scenario impacts the mobility of vehicles, and, in turn, the performance of applications and offloading systems. Thus, vehicular scenarios are of utmost importance for providing reliable metrics in experiments. Defining a vehicular scenario is also important to evaluate models that can be used in specific situations. In this way, computation offloading algorithms can be designed to adapt more efficiently to different scenarios. The following paragraphs present the description of the most used vehicular scenarios in computation offloading works in VEC: urban and highway.

Highway

Highway scenarios are generally characterized by a single road (with one or more lanes in each direction), one-dimensional vehicular mobility, high-speed vehicles, and few obstacles. In this scenario, wireless connections between vehicles are relatively stable if they travel in the same lane or the same direction (e.g., platooning). On the other hand, if vehicles travel in opposite directions, the wireless connections between them are short-lived and suffer interruptions and instability (SOMMER; DRESSLER, 2014; LI *et al.*, 2018). Continuous connectivity or coverage is also a significant challenge in highway scenarios due to the high cost

of deployment of RSUs or BSs (ASLAM; ZOU, 2011).

The highway scenario was considered in the experiments of several papers (LIU *et al.*, 2020; CHEN *et al.*, 2020; SUN *et al.*, 2019a). In Sun *et al.* (2019b), the scenario used consists of a 12 km segment of a Beijing highway with two lanes and two ramps. Qiao *et al.* (2018) carried out the experiments of its proposal on a four-lane two-way road.

Urban

Urban areas are regions with different street and avenue layouts. Two-dimensional vehicular mobility and many segments with intersections make this scenario more complex in terms of communication and routing decisions. Besides that, obstacles such as trees, buildings, viaducts, and tunnels are also complicating factors in this regard (ZHU *et al.*, 2016). A scenario widely used as an urban environment is the Manhattan mobility model, which has streets organized in the form of a grid (VIRIYASITAVAT *et al.*, 2011).

In carrying out the experiments, Rahman *et al.* (2020) considered an urban scenario with all bidirectional roads and multiple intersections. An urban region of Luxembourg with an area of 800 m X 600 m was used in the experiments of Feng *et al.* (2018). In other works, the authors used both the urban and the highway scenarios in evaluating the proposed solutions (ZHANG *et al.*, 2020; FENG *et al.*, 2017).

3.3.2 Vehicular Density

VANETs rely heavily on having vehicles nearby to exchange information and messages, especially on networks without infrastructure (SANGUESA *et al.*, 2016). Therefore, density is an important factor that impacts offloading processes in vehicular environments. Thus, in the following paragraphs, we describe the types of vehicular density considered in the computation offloading works in VEC.

Low

Low vehicular density scenarios, also called sparse scenarios, have few vehicles to exchange information with each other and maintain good network connectivity. For example, a rural road with very little vehicular traffic may not even have adequate connectivity between the few network devices. Thus, low densities can cause loss of messages and network packets due to

reduced communication capabilities (SANGUESA *et al.*, 2016). This type of vehicular density is the most used in offloading experiments. Its number of vehicles is around 11 vehicles/km on a highway (density by road length) and 25 vehicles/km^2 in an urban region (density by area) (SOUZA *et al.*, 2020; AKHTAR *et al.*, 2014; SANGUESA *et al.*, 2016).

In Qiao *et al.* (2018), the authors considered only low vehicular density as the number of vehicles varies from 1 to 20. The number of service vehicles used in the experiments of computation offloading of Chen *et al.* (2020) ranged from 3 to 30.

Medium

Medium vehicular density scenarios are intermediate scenarios between low and high density scenarios. They have a larger number of network nodes, with better connectivity, and generally without traffic jams. The number of vehicles at this density varies around 55 vehicles/km on highways and 120 vehicles/km^2 in urban regions (SANGUESA *et al.*, 2016; AKHTAR *et al.*, 2014).

This approximate density has been used in experiments in some offloading computation works. For example, Feng *et al.* (2018) considered an approximate rate between 88 to 97 vehicles/km^2 in an urban region. Sun *et al.* (2019a) considered 50 vehicles/km in a highway.

High

High vehicular density scenarios have better connectivity because more vehicles in the network are more likely to be within the communication range of others. However, these scenarios suffer from traffic jams, mainly during peak hours. There are many exchanging data vehicles at these times, generating simultaneous forwarding, broadcast storms, network packet collisions, and contention at MAC and physical layers. As a result, these problems congest the network, reduce message delivery, and make offloading processes more difficult. The variation in the number of vehicles ranges from approximately 120 vehicles/km on highways to 250 vehicles/km^2 in urban scenarios (SANGUESA *et al.*, 2016; AKHTAR *et al.*, 2014).

According to Souza *et al.* (2020), high is the least used vehicular density in the offloading experiments. None of the studies analyzed on computation offloading in VEC by other authors used this density. However, other computation offloading works used high density in other contexts (ZHANG *et al.*, 2019; GUO *et al.*, 2018).

3.3.3 Discussion

Even using different technologies and strategies, a computation offloading solution can work well in one scenario and not work in another. For example, scenarios with many intersections of streets (urban) or a single road (highway) influence network connectivity and interfere with the proposed solutions' performance. This also occurs with different vehicular densities (SOUZA *et al.*, 2020). Accordingly, it is important to submit the proposed solutions to different scenarios and densities, providing credibility to them, and evaluating the impact of specific situations. Despite this, none of these previous works used all three vehicular density types, and only two of them (ZHANG *et al.*, 2020; FENG *et al.*, 2017) used both the highway and urban scenarios in the experiments. On the other hand, our proposed framework and algorithms were submitted to evaluations under all types of scenarios and vehicular densities.

A contextual information that helps deal with different scenarios is the known routes of vehicles, a set of geographical coordinates that vehicles will travel. This information helps estimate vehicles' positions within a given time more accurately, avoiding sending tasks to those who will lose connectivity (SHIN *et al.*, 2020). Although the works mentioned in this chapter use various contextual information, none of them use the known routes of vehicles. Only two of our three proposed algorithms use this information: the GTT and the BCV.

Table 3 shows the main aspects discussed in this section.

Table 3 – Summary of experiment aspects of related works.

Reference	**Scenario**		**Vehicular Density**			**Contextual Information**
	Urban	**Highway**	**High**	**Medium**	**Low**	**Known Routes**
(LI *et al.*, 2014)						
(HOU *et al.*, 2016)	√				√	
(FENG *et al.*, 2017)	√	√		√	√	
(LIN *et al.*, 2017)						
(NABI *et al.*, 2017)	√			√	√	
(QIAO *et al.*, 2018)		√			√	
(WANG *et al.*, 2018)		√				
(NING *et al.*, 2018)	√					
(FENG *et al.*, 2018)	√			√	√	
(SUN *et al.*, 2019b)		√				
(SUN *et al.*, 2019a)		√		√	√	
(RAHMAN *et al.*, 2020)	√					
(CHEN *et al.*, 2020)		√			√	
(LIU *et al.*, 2020)		√			√	
(ZHANG *et al.*, 2020)	√	√				
GCF	√	√	√	√	√	
GTT	√	√	√	√	√	√
BCV	√	√	√	√	√	√

Source: The Author.

3.4 Concluding Remarks

According to the proposed taxonomy, this chapter presented the main related works of computation offloading from a vehicle in VEC to reduce application execution time. We saw that current solutions do not take full advantage of the technological potential in terms of communications, servers, and failures recovery mechanisms. Furthermore, these solutions do not consider all available data (such as CPU availability and known routes of vehicles) and do not use the most appropriate techniques for the problem at hand. Two aspects of communication standards in VEC were examined: technology and server type. We used these two aspects to compare and discuss our proposal with literature works. We then described the problem being treated and discussed the strategies used to solve it. Finally, we showed the scenarios and vehicular densities used in the experiments. Then, we commented on the importance of validating computation offloading solutions in different contexts and using other information available in task distribution processes.

4 SYSTEM MODEL AND PROBLEM FORMULATION

This chapter presents the modeling of the considered system. For example, the chapter explains the functioning of the analyzed network, the calculations of vehicular mobility, the transmission and processing of data, and the model of energy of the system. Also, we formulate the realization of computation offloading in vehicular edge computing as an optimization problem.

A special contribution of this chapter is the calculation of vehicular position prediction based on known routes of vehicles. This contextual information helps to predict vehicle positioning more accurately and avoid offloading failures. To the best of our knowledge, it is the first time that a computation offloading work in vehicular networks uses this information.

This chapter is organized into three sections. Section 4.1 presents the modeling of the system. Section 4.2 describes the formulation of the problem that we propose to solve. Finally, Section 4.3 highlights the final considerations of the chapter.

4.1 System Model

This section presents an overview of the system model. First, Section 4.1.1 describes the network general structure. Then, Sections 4.1.2 and 4.1.3 present the models for communication and computation, respectively. At last, Section 4.1.4 presents the energy model. In turn, Table 4 lists the most used notations.

4.1.1 Network General Structure

The network topology considered in this work has different types of nodes. For example, it has a set of vehicles and a set of edge servers. Since each edge server is connected to a different 5G base station, the total number of edge servers present on the network also represents the total number of 5G base stations. The 5G base stations are installed on the shoulders of streets or roads. The edge servers are connected via optical fiber to the 5G base stations. Vehicles can simultaneously process, store, transmit data, and use sensors. Besides, vehicles periodically generate beacon messages and distribute them in a one-hop broadcast to all nodes within their communication range.

We consider *client* as a vehicle that offloads tasks to other vehicles (via V2V communication) or to edge servers (via V2I communication). Offloading decisions are made based

Table 4 – Most used notations.

Notation	Definition
client	Client vehicle
server	Chosen server for the *client*
R	Communication range
t	Time
d	Distance
r	Transmission rate
B	Bandwidth
s	Data size
C	Computational capacity
n_τ	Total number of tasks in a workload
τ	Task
c	CPU cycles required to process a task
W	Number of tasks to be executed by *client*
X	Number of tasks to be executed by the *server*
E_{cur}	Current stored energy level
A	Accumulated minimum energy level

Source: The Author.

on information from other devices. However, the decision of whether and how to offload is made only by the *client*. This decision can lead *client* to take the following actions: execute all tasks locally, execute all tasks remotely (on one or more servers), or execute some tasks locally and some more remotely.

A *server* is any vehicle or edge server that can receive and process one or more tasks sent by the *client*. All vehicles and edge servers in our topology can act as a server. Such possible servers continuously run offloading services in background that can 1) advertise the computational resources available on the server to other devices and 2) enable them to receive and execute tasks from other devices.

Table 5 presents the interfaces of the network nodes and the types of nodes they can connect. The *client* has two network interfaces in our network topology: WAVE/IEEE 802.11p and 5G/mmWave. All other vehicles are servers only and have only WAVE/IEEE 802.11p interfaces. All edge servers have only one connection to base stations (which also have 5G/mmWave interfaces). All vehicles present the same communication range. All base stations connected to the edge servers also have the same communication range. For simplicity, we only consider one-hop communications in the WAVE/IEEE 802.11p interface. On the 5G/mmWave interface, if the *client* wants to transmit something to the edge server, it must first transmit to the 5G base station. The latter then forwards the data to the edge server via a wired link. If the edge server wants to transmit something to a vehicle, the reverse path is taken.

Table 5 – Network node interfaces.

Node	Interface	To
Client vehicle	WAVE/IEEE 802.11p	Other vehicles
	5G/mmWave	Base station
Server vehicle	WAVE/IEEE 802.11p	Other vehicles
Base station	5G/mmWave	*Client* vehicle
	Wired	Edge server
Edge server	Wired	Base station

Source: The Author.

4.1.2 Communication Model

This section presents the communication models related to the link lifetime, position prediction based on known routes of vehicles, and data transmission rate.

Link Lifetime

It is possible to estimate how long two neighboring network nodes remain connected within each other's communication range. The estimate is made through kinematic calculations since parameters such as speed, direction, and distance between nodes do not vary significantly (SOUZA *et al.*, 2013). Härri *et al.* (2008) proposed a way to calculate this estimate. For this, we use four parameters: p_x position of the node on the x-axis, p_y position of the node on the y-axis, v_x vector velocity of the node on the x-axis, and v_y vector velocity of the node on the y-axis. We assume that the nodes follow a linear path in a short period. Thus, Equation 4.1 describes the position of the node i as a function of time t:

$$p_i(t) = \begin{bmatrix} p_{x_i} + v_{x_i} \cdot t \\ p_{y_i} + v_{y_i} \cdot t \end{bmatrix}, \tag{4.1}$$

where $p_i(t)$ is the position of node i at time t.

We consider that a node j is a neighbor of the node i. Thus, the Equation 4.2 shows how to estimate the future distance between nodes i and j:

$$\begin{aligned} d_{i,j}^2 = d_{j,i}^2 = \|p_j(t) - p_i(t)\|^2 &= \left(\begin{bmatrix} p_{x_j} - p_{x_i} \\ p_{y_j} - p_{y_i} \end{bmatrix} + \begin{bmatrix} v_{x_j} - v_{x_i} \\ v_{y_j} - v_{y_i} \end{bmatrix} \cdot t \right)^2 \\ &= \alpha_{i,j} t^2 + \beta_{i,j} t + \gamma_{i,j}, \end{aligned} \tag{4.2}$$

where $\alpha_{i,j} \geq 0$, $\gamma_{i,j} \geq 0$. Then, the future relative distance between j and i is $d_{i,j}(t) = \sqrt{\alpha_{i,j}t^2 + \beta_{i,j}t + \gamma_{i,j}}$.

With this, it is possible to calculate the link lifetime. According to Härri *et al.* (2008) and Menouar *et al.* (2007), the link lifetime of the two nodes (i and j) is the estimated time $t_{link_{i,j}} = t_1 - t_0$, where t_1 is the time when the distance d between the two nodes becomes greater than their communication range (R) and t_0 is the initial time of the nodes. For the two nodes to be connected, d must be less than or equal to R.

For $t_{link_{i,j}}$ to be calculated, it is necessary to make $d_{i,j}(t) \leq R$. If we square both sides of the inequality and put R on the other side, we get the Equation 4.3, which gives the value of $t_{link_{i,j}}$:

$$d_{i,j}^2(t) - R^2 = 0 \tag{4.3}$$

Since we already have an equation that describes $d_{i,j}^2(t)$, we get the Equation 4.4 below:

$$\alpha_{i,j}t^2 + \beta_{i,j}t + \gamma_{i,j} - R^2 = 0, \tag{4.4}$$

where t represents $t_{link_{i,j}}$. Thus, to know the value of $t_{link_{i,j}}$, it is only necessary to solve the second degree equation presented in Equation 4.4.

However, if two nodes have very similar mobility (for example, if they are close to each other, with similar velocities and going in the same direction), t_{link} tends to be infinite. To adjust this question, Namboodiri and Gao (2007) proposed an upper limit constant for very large values of t_{link}. We defined this constant as $t_{maxlifetime}$ with a value of 100 seconds. That way, if $t_{link} > 100$, t_{link} becomes 100 seconds.

Position Prediction Based on Known Routes of Vehicles

Public or private vehicles can share information from their on-board navigation systems, such as their trajectories/routes and the estimated arrival time to the destination. This sharing is done through the exchange of messages between devices. With this, it is possible to estimate, with more precision, if two neighboring network nodes will still be connected after a given time $t_{\varkappa}$ (SHIN *et al.*, 2020).

The trajectory or route of a vehicle j is a set of points (geographic coordinates) that will be traveled by it as follows: $[(p_{x_j}(t_0), p_{y_j}(t_0))$; $(p_{x_j}(t_1), p_{y_j}(t_1))$; ... ; $(p_{x_j}(t_n), p_{y_j}(t_n))]$, where $(p_{x_j}(t_0), p_{y_j}(t_0))$ represents the latitude and longitude of the vehicle at the initial time t_0 and $(p_{x_j}(t_n), p_{y_j}(t_n))$ represents the latitude and longitude of the vehicle at the estimated arrival time t_n.

Thus, the distance to be travelled (d_{total}) is calculated as follows:

$$d_{total} = \sum_{\ell=1}^{n} \sqrt{(p_{x_j}(t_\ell) - p_{x_j}(t_{\ell-1}))^2 + (p_{y_j}(t_\ell) - p_{y_j}(t_{\ell-1}))^2} \ . \tag{4.5}$$

Therefore, the average vehicle speed is:

$$\overline{v_j} = \frac{d_{total}}{(t_n - t_0)} \ . \tag{4.6}$$

Then, the estimated distance traveled by the vehicle j in a given time $t_\varkappa$ is:

$$d_{est} = \overline{v_j}(t_\varkappa - t_0) \ . \tag{4.7}$$

For simplicity, we call the estimated position $p_j(t_\varkappa)$ of $(\mathfrak{p}_x, \mathfrak{p}_y)$. If $t_\varkappa > t_n$, then $(\mathfrak{p}_x, \mathfrak{p}_y)$ it is the final position of the trajectory, already known. If $t_\varkappa < t_n$, to find $(\mathfrak{p}_x, \mathfrak{p}_y)$, add the distances between the points of the vehicle's trajectory, from the initial point as in Equation 4.5, until the sum of these distances is greater than d_{est}. When this happens, we know that $(\mathfrak{p}_x, \mathfrak{p}_y)$ is approximately on the line between the last and the penultimate points added, respectively (a_x, a_y) and (b_x, b_y). With these two points, we calculate the general equation of the line as a function of $(\mathfrak{p}_x, \mathfrak{p}_y)$ as follows:

$$(a_y - b_y)\mathfrak{p}_x + (a_x - b_x)\mathfrak{p}_y + a_x b_y - b_x - a_y = 0 \ . \tag{4.8}$$

In addition, the distance between $(\mathfrak{p}_x, \mathfrak{p}_y)$ and (b_x, b_y) is d_{est} minus the distance from the initial point of the vehicle's trajectory until (b_x, b_y), which we call d_ρ. Thus, we can obtain another equation as a function of $(\mathfrak{p}_x, \mathfrak{p}_y)$:

$$\sqrt{(\mathfrak{p}_x - b_x)^2 + (\mathfrak{p}_y - b_y)^2} = d_{est} - d_\rho \ . \tag{4.9}$$

Combining the Equations 4.8 and 4.9, we were able to obtain the estimated position of the vehicle j in time $\varkappa$. The same procedures are used to calculate the position of a vehicle i in time $\varkappa$. After this, we calculate the distance between i and j in time $\varkappa$ and check if it is less than the communication range.

Data Transmission Rate

According to Raza *et al.* (2020), Wang *et al.* (2018), Zhang *et al.* (2019), and Sun *et al.* (2018), in WAVE/V2V communications, the data transmission rate between a node i and a node j at a given time t is given by:

$$r_{V2V_{i,j}}(t) = B_{V2V} \log_2(1 + \frac{P_t L_{V2V} |\phi^2|}{\omega}), \tag{4.10}$$

where B_{V2V} is the bandwidth between i and j, P_t is the transmission power of the node, L_{V2V} is the loss of system propagation, ϕ is the fading coefficient of the uplink channel, and ω is the power of the white Gaussian noise.

Thus, the average uplink rate between nodes i and j is given by:

$$\overline{r_{V2V_{i,j}}} = \frac{\int_0^{t_{link_{i,j}}} r_{V2V_{i,j}}(t)\ \mathrm{d}t}{t_{link_{i,j}}} \quad . \tag{4.11}$$

Conforming to Chen *et al.* (2020), for simplicity, we can neglect the time to access the control (CCH) and service (SCH) WAVE channels and the time spent on switching channels, such as the guard interval.

Thus, the time it takes to transmit data of size s from i to j via WAVE/V2V is given by:

$$t_{trans_{i,j}} = \frac{s}{\overline{r_{V2V_{i,j}}}} + \frac{d_{i,j}}{v_{prop}} + t_{other}, \tag{4.12}$$

where v_{prop} is the propagation speed over the wireless medium and t_{other} are estimates of possible network and queue delays.

For 5G/V2I communications, Shaham *et al.* (2019), Giordani *et al.* (2018), and Cui *et al.* (2019) show that the data transmission rate between a node i and a node j in a given time t is given by:

$$r_{V2I_{i,j}}(t) = B_{V2I}\log_2(1+\frac{P_t d_{i,j}^{-\varepsilon}|\phi^2|}{\omega}), \tag{4.13}$$

where B_{V2I} represents the bandwidth between i and j, $d_{i,j}$ is the distance between i and j and the factor ε is the exponent of propagation loss of the system. The transmission delay of the wired link is neglected in this work (CUI *et al.*, 2019). This negligence is due to the fast transmission rates on this type of link. Also, the wired links typically are a few meters long, as each base station has an edge server coupled.

In this way, the average uplink rate between nodes i and j is given by:

$$\overline{r_{V2I_{i,j}}} = \frac{\int_0^{t_{link_{i,j}}} r_{V2I_{i,j}}(t)\ \mathrm{d}t}{t_{link_{i,j}}}\ . \tag{4.14}$$

Thus, the time it takes to transmit data of size s from i to j, via 5G/V2I, is given by:

$$t_{trans_{i,j}} = \frac{s}{\overline{r_{V2I_{i,j}}}} + \frac{d_{i,j}}{v_{prop}} + t_{other}\ . \tag{4.15}$$

4.1.3 Computation Model

Each node on the network, vehicle or edge server, has different computational capacities and CPU availability (which varies over time). Each CPU available from any node on the network can only execute one computation task at a time. Tasks are placed in a queue and taken out in First In, First Out (FIFO) model to be executed.

We consider a computationally intensive and real-time application that generates a workload or set $T = \{\tau_1, \tau_2, \tau_3, ..., \tau_{n_\tau}\}$ of computation tasks. Each task $\tau \in T$ can be processed locally by the *client* or remotely in an independent, asynchronous, and parallel way. Thus, each workload can have its tasks distributed for local or remote execution (on the edge servers or in other vehicles) or in both local and remote environments. In addition, each task $\tau \in T$ is a tuple composed of the following parameters $\{c_\tau, s_{\tau,up}, s_{\tau,down}\}$, where c_τ indicates the total

number of CPU cycles required to execute the task τ, $s_{\tau,up}$ shows the data size for upload of τ (which includes input parameters, the code to be executed and information about the device that generated the task) and $s_{\tau,down}$ shows the size of the processing results of τ for download (which includes information on which device the results should be sent to). Each task is executed with 100% of the CPU available for task executions.

Next, we can see the details of the computational modeling of the execution for each environment (RAZA *et al.*, 2020).

Local Computation

When the *client* chooses to execute a task locally, the local execution delay of the *client* is set to t_{client}. C_{client} is described as the computational capacity of the *client* node (in CPU cycles per second). t_{client} consists of two parts: 1) queue delay (there may be other tasks waiting to be executed or in execution) and 2) processing delay of the task on the CPU. The queue delay is given by:

$$t_{\substack{client,\\queue}} = \sum_{w=1}^{g} \frac{c_w}{C_{client}} , \tag{4.16}$$

where w represents each task in the queue of the *client* node and g represents the number of tasks that were already waiting in the queue or executing.

The processing delay of a task τ on the *client* CPU is given by:

$$t_{\substack{client,\\proc}} = \frac{c_\tau}{C_{client}} . \tag{4.17}$$

Thus, the local execution delay in the *client* node of a task $\tau \in T$ is given by:

$$t_{client} = t_{\substack{client,\\queue}} + t_{\substack{client,\\proc}} . \tag{4.18}$$

Remote Computation

A *client* vehicle generates a task and send it to execute on a remote server. As part of vehicular edge computing, the *server* is a vehicle or edge server. Such a *server* executes the task and returns the result of the execution to the *client* vehicle that generated the task.

Thus, the delay in executing the task on the *server* is divided into four parts. The first part is the upload of $s_{\tau,up}$ from the *client* vehicle to the *server*, $t_{\underset{up}{client},}$. If the upload is via WAVE/V2V, the delay is given by Equation 4.12. If the upload is via 5G/V2I, the delay is given by Equation 4.15.

The second part is the waiting time for the task in the *server* queue $t_{\underset{queue}{server},}$ (MIDYA *et al.*, 2018; RAHMAN *et al.*, 2020). To be processed, the task must wait for all tasks that were already waiting or executing on the server to finish executing. This time is given by:

$$t_{\underset{queue}{server},} = \sum_{x=1}^{q} \frac{c_x}{C_{server}} \ , \tag{4.19}$$

where x represents each task in the queue of the *server* node and q represents the number of tasks that were already waiting in the queue or in execution.

The third part consists of the time it takes to process a task τ:

$$t_{\underset{proc}{server},} = \frac{c_\tau}{C_{server}} \ . \tag{4.20}$$

The fourth part of the delay is the time it takes to transmit the processing result ($s_{\tau,down}$) from the *server* to the *client*, $t_{\underset{down}{client},}$. If the transmission is via WAVE/V2V, the time is given by Equation 4.12. If it is via 5G/V2I, the time is given by Equation 4.15.

Thus, the delay for executing a task $\tau \in T$ on a remote server is given by:

$$t_{server} = t_{\underset{up}{client},} + t_{\underset{queue}{server},} + t_{\underset{proc}{server},} + t_{\underset{down}{client},} \ . \tag{4.21}$$

Total Execution Time

As we consider that workload has a total of n_τ tasks, they are distributed to be executed locally and on remote servers (vehicles or edge servers).

So, we assume that tasks are distributed as follows:

- W tasks are distributed to the *client*;
- X_1 tasks are distributed to the $server_1$, X_2 to the $server_2$, X_3 to the $server_3$ and so on up to X_k to the $server_k$, so that $X_1 + X_2 + X_3 + ... + X_k = X$;

Thus, the total time required to execute tasks locally is given by:

$$t_{\substack{client,\\total}} = \sum_{w=1}^{W} t_{\substack{client,\\w}} \quad , \tag{4.22}$$

where w represents each task distributed to the $client$ node.

In turn, the total time required to execute tasks on remote servers is given by:

$$t_{\substack{servers,\\total}} = max\left\{ \sum_{x_1=1}^{X_1} t_{\substack{server_1,\\x_1}} \, , \sum_{x_2=1}^{X_2} t_{\substack{server_2,\\x_2}} \, , \ldots , \sum_{x_k=1}^{X_k} t_{\substack{server_k,\\x_k}} \right\} , \tag{4.23}$$

where x_1 represents each task distributed to the $server_1$, x_2 each task to the $server_2$, x_k each task to the $server_k$.

Therefore, the total time to execute the workload is described as:

$$t_{total} = max\left\{ t_{\substack{client,\\total}} \, , \, t_{\substack{servers,\\total}} \right\} . \tag{4.24}$$

4.1.4 Energy Model

Regarding energy, the network nodes can be divided into four types: Combustion-powered Vehicle in Motion (CVM), Combustion-powered Vehicle Off (CVO), Electric Vehicle (EV), and edge server. Combustion-powered Vehicles (CVs) have internal combustion engines that generate energy by burning fuel. In CVMs, an alternator supplies power to the vehicle's electronic systems and recharges the battery. In CVOs, a battery with limited capacity provides power for electronic equipment (REIS *et al.*, 2018; REIS *et al.*, 2017; BOUKERCHE; SOTORO, 2020). EVs have engines and electronic equipment that run on the energy of rechargeable batteries with large but limited storage capacities. We consider that EVs that are turned off are not being charged. Finally, the edge server is a type of node that is connected to an electric power operator (HUANG *et al.*, 2020).

Thus, we consider that edge servers and CVMs have unlimited energy. CVOs and EVs, on the other hand, have a limited amount of stored energy E_{cur} and need to have an accumulated minimum energy level A to perform offloading tasks.

Besides, each network node, vehicle or edge server, has different energy consumption levels when processing tasks. According to Cui *et al.* (2019) and Huang *et al.* (2020), the energy consumed to execute a task τ can be formulated as follows:

$$E_{proc} = \frac{c_\tau}{C} P_{proc} \ , \tag{4.25}$$

where P_{proc} represents the circuit power related to a coefficient of the model, architecture, and capacitance of the CPU chip (this variable has different values for each network node).

The energy consumed in small packet transmissions and receptions (e.g., the result of a computation task and request and reply messages), in general, is neglected. However, uploading a task is taken into account (WANG *et al.*, 2019; LI *et al.*, 2020b; MISRA; BERA, 2019; GU; ZHANG, 2021; HUANG *et al.*, 2020). Thus, the energy consumed in the transmission of a task τ is formulated as follows:

$$E_{trans} = \frac{s_{\tau,up}}{\overline{r_{i,j}}} P_{trans} \ , \tag{4.26}$$

where P_{trans} is the transmission power (considered fixed for WAVE devices and fixed for 5G devices).

As seen in Section 4.1.3 and Equation 4.25, the energy consumed by the client is given by:

$$E_{\substack{client, \\ total}} = \sum_{w=1}^{W} \frac{c_w}{C_{client}} P_{\substack{client, \\ proc}} + \sum_{x_1=1}^{X_1} \frac{s_{x_1,up}}{\overline{r_{i,j}}} P_{\substack{client, \\ trans}} + ... + \sum_{x_k=1}^{X_k} \frac{s_{x_k,up}}{\overline{r_{i,j}}} P_{\substack{client, \\ trans}} \quad , \tag{4.27}$$

meaning the energy consumed when executing tasks locally and the energy consumed when transmitting tasks to be executed on remote servers.

The energy consumed when executing tasks on a certain remote server k is estimated as follows:

$$E_{\substack{server_k, \\ proc}} = \sum_{x_k=1}^{X_k} \frac{c_{x_k}}{C_{server_k}} P_{\substack{server_k, \\ proc}} \quad . \tag{4.28}$$

Consequently, the energy consumed when executing tasks on remote servers can be described as:

$$E_{\substack{servers,\\total}} = \sum_{x_1=1}^{X_1} \frac{c_{x_1}}{C_{server_1}} P_{\substack{server_1,\\proc}} + \sum_{x_2=1}^{X_2} \frac{c_{x_2}}{C_{server_2}} P_{\substack{server_2,\\proc}} + \ldots + \sum_{x_k=1}^{X_k} \frac{c_{x_k}}{C_{server_k}} P_{\substack{server_k,\\proc}} \quad . \tag{4.29}$$

Therefore, the total energy consumed in the offloading process is given by:

$$E_{total} = E_{\substack{client,\\total}} + E_{\substack{servers,\\total}} \, . \tag{4.30}$$

4.2 Problem Formulation

This section formulates the problem addressed in this thesis. The objective of this work is to minimize the execution time of a vehicular application through the computation offloading process in vehicular edge computing systems, satisfying reliability restrictions. In this process, an application running on the *client* vehicle distributes its computation tasks so that other devices execute them, return the results, and the execution time is reduced. So the problem can be formulated as follows:

$$\begin{aligned}
&P1 : min\ t_{total} \\
&\quad s.t.\ C1 : W + X = n_\tau \ , \\
&\qquad C2 : \sum_{x_k=0}^{X_k} t_{\substack{server_k,\\x_k}} < t_{link}{}_{\substack{client,\\server_k}} \\
&\qquad\quad \vee \ \left(d_{\substack{client,\\server_k}} \left(t_{\substack{server_k,\\x_k}} \right) < R \right) , \\
&\qquad C3 : E_{\substack{client,\\cur}} - E_{\substack{client,\\total}} > A_{client} \ , \\
&\qquad C4 : E_{\substack{server_k,\\cur}} - E_{\substack{server_k,\\proc}} > A_{server_k} \ .
\end{aligned}$$

Thus, problem P1 aims to minimize t_{total} (Equation 4.24). The C1 constraint ensures that the sum of the total tasks executed on the client (W) and the total tasks executed on remote servers (X) is equal to the number of tasks in the application workload (n_τ). The constraint C2

means that the time $t_{\underset{x_k}{server_k,}}$ to execute a workload x_k on a given $server_k$ needs to be less than the link lifetime between *client* and $server_k$. Or, if the *client* and $server_k$ routes are known, the distance between their estimated positions in time $t_{\underset{x_k}{server_k,}}$ must be less than the communication range R. With the constraint C2, the problem formulation aims to contribute to greater reliability of the computation offloading process, preventing *client* and a given $server_k$ from getting out of each other's communication range.

The constraint C3 states that the client's current energy level ($E_{\underset{cur}{client,}}$) minus its total energy consumption estimate for offloading ($E_{\underset{total}{client,}}$) does not reach its minimum energy level (A_{client}). The constraint C4 establishes the same as C3 but is related to any $server_k$. With constraints C3 and C4, it is ensured that energy-constrained network nodes do not have their operations interrupted due to a lack of energy caused by offloading processes.

This optimization problem of task assignment/scheduling with multiple constraints is considered a NP-hard problem. Thus, there is no polynomial-time algorithm that can find the optimal solution (ZHU *et al.*, 2018; FENG *et al.*, 2017; SUN *et al.*, 2018).

4.3 Concluding Remarks

In this chapter, the description of the network structure explained that the *client* vehicle has interfaces for WAVE and 5G networks. Through them, the *client* can access computational resources from other vehicles and edge servers. In the communication modeling, two ways of predicting vehicular mobility were presented: through the calculation of the link lifetime and prediction based on known routes of vehicles. In addition, it was shown how the data transmission rates and time estimates are calculated using the WAVE and 5G networks. Computation modeling explained what a workload and task are and how they are processed locally or remotely. This chapter also presented how to calculate the total execution time estimate for a workload. When modeling energy, different types of vehicles were considered. Moreover, the client and server energy consumption calculations when processing and transmitting tasks were presented. Finally, the formulation of the problem addressed in this thesis was described.

5 PROPOSED FRAMEWORK AND DECISION ALGORITHMS

This chapter presents a new framework and new decision algorithms for computation offloading in vehicular edge computing systems. The objective of these solutions is to minimize the execution time of vehicular applications, satisfying mobility and energy constraints. The framework supports a vehicle to discover computational resources and gather contextual data from devices within its communication range. Thus, using WAVE and 5G networks, a vehicle can send computation tasks to edge servers and nearby vehicles. Also, the framework assists the process of computation offloading to remote devices and provides a failure recovery mechanism.

Decision algorithms are the core of the framework. The function of these algorithms is to make the best possible distribution of application tasks. In this chapter, we propose three new algorithms. The first two are greedy task assignment algorithms (upload order does not matter) that use different strategies. The third and last is an intelligent task scheduling algorithm (upload order matters) based on the ABC metaheuristic.

The main contributions reported in this chapter are:

- A context-oriented framework for computation offloading in vehicular edge computing with descriptions of the conceptual architecture and offloading and failure recovery processes.
- Simultaneous use of WAVE and 5G technologies, combining their advantages, increasing capacities, and decreasing task transmission delays.
- Use of a special contextual information about complete routes of vehicles, providing more reliability and accuracy for the computation offloading processes.
- Greedy for CPU Free (GCF), a task assignment algorithm that prioritizes computing offloading to servers with the lowest queue times and the shortest distances for the client.
- Greedy Task by Task (GTT), a decision algorithm that seeks to assign each application task to the best possible server, considering CPU capacity, queue time, distance to the client, and known routes of vehicles (if available).
- ABC for Computation Offloading in VEC (BCV), an intelligent task scheduling algorithm inspired by bees searching for food sources or solutions, which are evaluated according to the estimate of the total execution time of the workload.

The remainder of this chapter is organized as follows. Section 5.1 describes the proposed framework. The proposed decision algorithms are detailed in Section 5.2. Finally, Section 5.3 outlines the concluding remarks.

5.1 Proposed Framework

This section describes the details of the proposed framework. Section 5.1.1 presents the conceptual architecture of the computation offloading framework for vehicular systems. In turn, Section 5.1.2 explains the flow of the computation offloading process managed by the framework.

5.1.1 Framework Architecture

Figure 8 shows the *Application* and *Partitioner* modules and conceptual architecture of the proposed framework. The box with dotted lines represents an application running on a vehicle, along with the *Partitioner* module and the proposed framework. The latter is represented by the smaller box with a gray background. The arrows indicate the direction of the information flow between the modules.

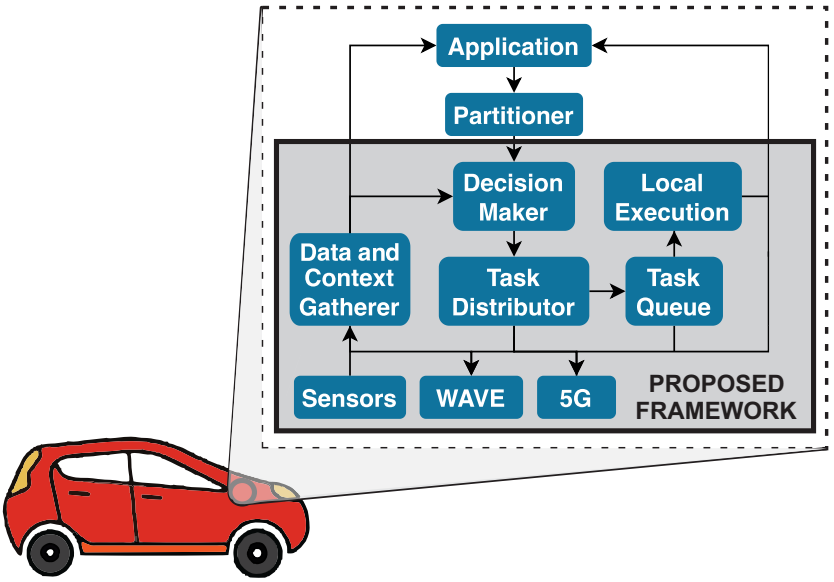
The *Application* module represents the applications running on the vehicle's operating system. As seen in Figure 8, it sends data to a *Partitioner* module in order to analyze whether the application workload can be partitioned. If it is possible, the *Partitioner* divides the application workload into smaller tasks that can be executed on different devices and in a parallel, asynchronous, and independent way. The application workload then moves to the *Decision Maker* module, which decides where each task should execute and reports the decision to the *Task Distributor* module. The latter distributes tasks for local or remote execution. After the workload has been processed, the *Application* receives the results through the *Local Execution* or the *Data and Context Gatherer* module, when the results come from remote devices. The *Application* also receives information from local sensors and other devices. This information is captured through the *Data and Context Gatherer* module.

Next, we describe each module of the proposed framework.

Sensors

The *Sensors* module is responsible for sending to the *Data and Context Gatherer* module local information such as location (via Global Positioning System), energy, speed, and direction.

Figure 8 – *Application* and *Partitioner* modules and architecture of the framework.



Source: The Author.

Task Queue

Tasks from remote devices or the client are placed in the *Task Queue* by the *Task Distributor* module. After a task passes through the queue, it goes to the *Local Execution* module. Information about the storage capacity of the queue and the number of tasks in it is periodically passed on to the *Data and Context Gatherer* module.

Local Execution

With CPU and other resources, the *Local Execution* module processes tasks that come from the *Task Queue* with the processing delays described in Section 4.1.3. Through information contained in the task, this module checks whether it is local or came from a remote device. Then, it can forward the processing result to the local application or to remote devices (via the *WAVE* or *5G* modules).

WAVE and 5G

The *WAVE* module is responsible for sending and receiving messages from the WAVE network via V2V. The *5G* module is responsible for sending and receiving messages from the 5G network via V2I.

These modules can send periodic signaling and safety messages and other data to remote devices. Upon receiving messages, these modules forward them to the *Data and*

Context Gatherer module. These messages can be: requests to execute a task to a remote device (offloading request), replies to offloading requests (offloading reply), tasks, data downloaded to a running application, and context data from other devices.

Task Distributor

This module analyzes the tasks received from the *Decision Maker* and the location where they will be executed. If the decision is to execute a task locally, the *Distributor* forwards it to the local *Task Queue*. If the decision is to run a task remotely, the *Distributor* sends it to the *WAVE* or *5G* modules (or both), depending on the choice made by the *Decision Maker*.

This module is also responsible for storing a backup of tasks that have been offloaded to remote servers. Thus, upon being informed that the remote server has failed to return the processing result, this module then sends the stored copies of the lost tasks to be executed locally.

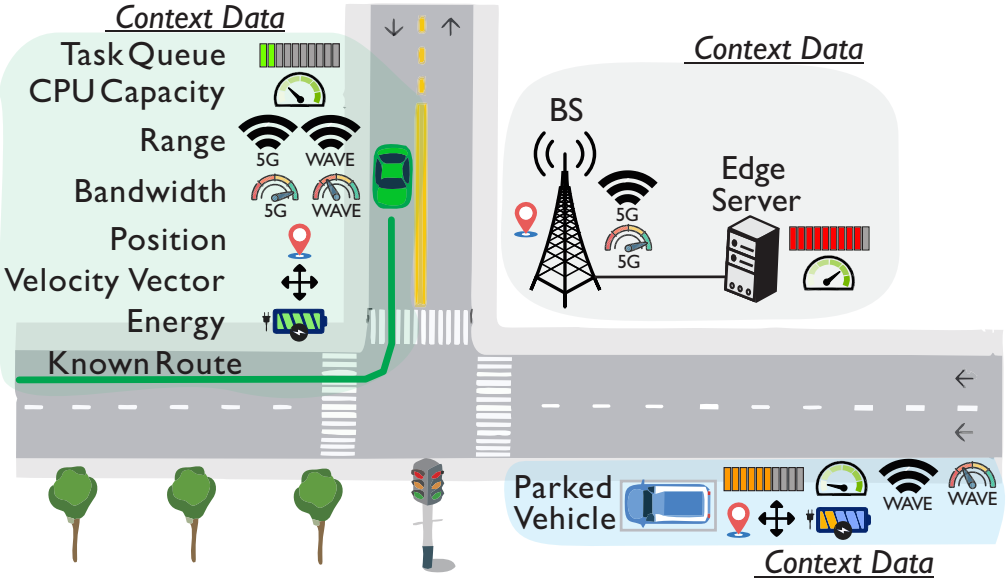
Data and Context Gatherer

Through the *WAVE* and *5G* modules, the *Data and Context Gatherer* can receive messages involving application data or remote processing results. Then, the *Gatherer* forwards it to the local *Application* module. Upon receiving an offloading request, a reply for an offloading request, a task to be executed, or remote contextual information, this module forwards them to the *Decision Maker* module. If the *Gatherer* receives periodic local contextual information through the *Sensors*, the *Task Queue* and the *Local Execution* module (CPU capacity), this module forwards it to the *Application* module or to the *Decision* module.

The *Data and Context Gatherer* module captures contextual information about other network nodes via request/reply processes through the *WAVE* and *5G* modules. As shown in Figure 9, this information can be values of position, bandwidth and range (WAVE and 5G), task queue time, and CPU capacity. If the node is a vehicle, this module also captures the velocity vector, the current stored energy, and the complete route to the destination (if available).

This module also monitors possible offloading failures. This is done by analyzing the signaling messages received from the chosen remote servers and verifying their connectivity with the client. After failure detection, the *Gatherer* notifies the *Decision Maker* that triggers the *Task Distributor*. More details of failure handling are described in Section 5.1.2.

Figure 9 – Contextual data of network nodes in a VEC system.



Source: The Author.

Decision Maker

Upon receiving tasks from the *Partitioner*, the *Decision Maker* module calculates the estimated energy consumption for the following three hypothetical cases: the *client* (1) executes all workload tasks locally, (2) sends all workload tasks via V2I / 5G, and (3) sends all workload tasks via V2V / WAVE. Suppose the *client* has enough energy to perform these three cases, preventing the constraint C3 from being breached. In that case, the *Decision Maker* module starts resources discovery. It gathers all contextual information (local and remote) from the *Data and Context Gatherer*. Then, the *Decision Maker* module calculates additional information such as distances between devices, transmission and processing time, and estimates of link lifetime and connectivity.

With all this information, this module assigns, through a decision algorithm (see Section 5.2), tasks for the chosen devices to execute. Subsequently, it informs the task assignment/scheduling (depending on whether the order of sending tasks matters) decision to the *Task Distributor* module.

This module also receives other types of messages. For example, when receiving offloading requests, the *Decision Maker* analyzes whether the vehicle's current energy level is above its minimum level ($E_{cur} > A$). If so, it triggers the *Distributor* to reply to the requesting device agreeing to process its tasks. Upon receiving a positive reply for an offloading request, it adds the device to the list of possible servers to make the decision. If the device is chosen as a

server, this module sends the tasks to the remote device through the *Distributor* and *WAVE* or *5G* module. Upon receiving a task to be executed, it forwards it to the local *Task Queue*, through the *Distributor* module.

5.1.2 Computation Offloading Process

This section presents the computation offloading process managed by the proposed framework. The process consists of four main parts: (1) Resources discovery, (2) Offloading decision, (3) Send/receive tasks, and (4) Failure recovery.

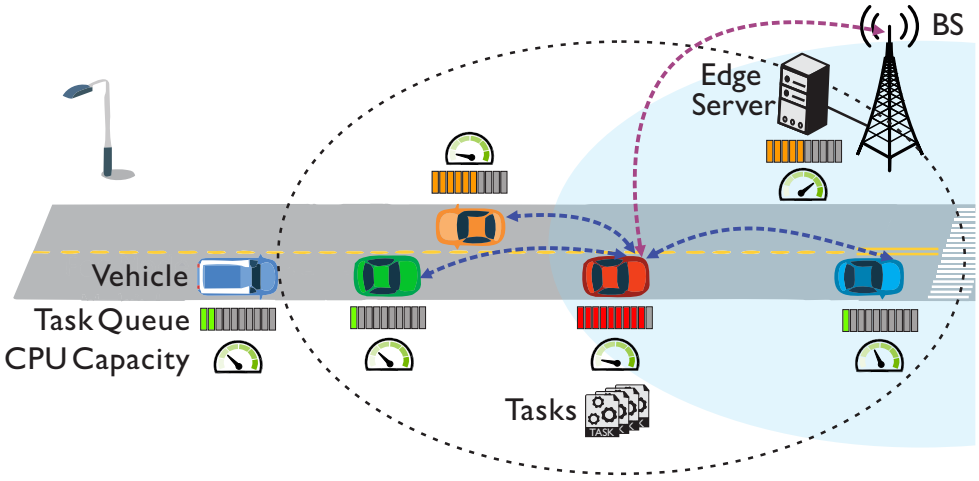
Resources Discovery

The computation offloading process starts when the *Decision Maker* module receives tasks from the *Application*/*Partitioner*. Then, through the *Distributor*, *WAVE* and *5G* modules, it triggers the resources discovery.

At this time, as shown in Figure 10, the client (red) sends a one-hop WAVE request via broadcast to the other vehicles and sends a 5G request via unicast to the edge server. These requests are used to gather contextual information from possible servers. The WAVE request is made via V2V connections (blue dotted lines) within the limits of the communication range of the technology (black dotted line). The 5G request first passes through the 5G base station via V2I connection (purple dotted line), limited by the 5G communication range (light blue background). In this case, the client has four tasks to be executed, its computational capacity is low (pointer almost at least), and its task queue for execution is almost full (bar with small red blocks).

After the client´s first contact, the possible server analyzes whether it has enough energy ($E_{cur} > A$). If it has, it replies with a tuple containing its location, data rate and communication range (WAVE or 5G), CPU capacity, task queue condition, and information about energy (current and minimum level and circuit power related to the CPU chip). If this possible server is a vehicle, it also sends its speed, direction, and, if available, its estimated arrival time to the destination and its complete route. This reply reaches the client via the *WAVE* and *5G* modules, which forwards it to the *Data and Context Gatherer* module, which in turn sends it to the *Decision* module. Subsequent to waiting a constant time of ψ milliseconds, the client discovers possible servers and proceeds to the task assignment/scheduling decision.

Figure 10 – Discovery of computational resources.



Source: The Author.

Decision

After congregating all the necessary information, the *Decision Maker* module executes an algorithm (see Section 5.2) to decide the assignment/scheduling of tasks. In this step, the decision considers the objective, restrictions, and contextual information to make a good decision, aiming to solve the problem P1. It must assign each task to a server (local or remote) to execute and choose the communication technology to be used (WAVE or 5G, or both simultaneously).

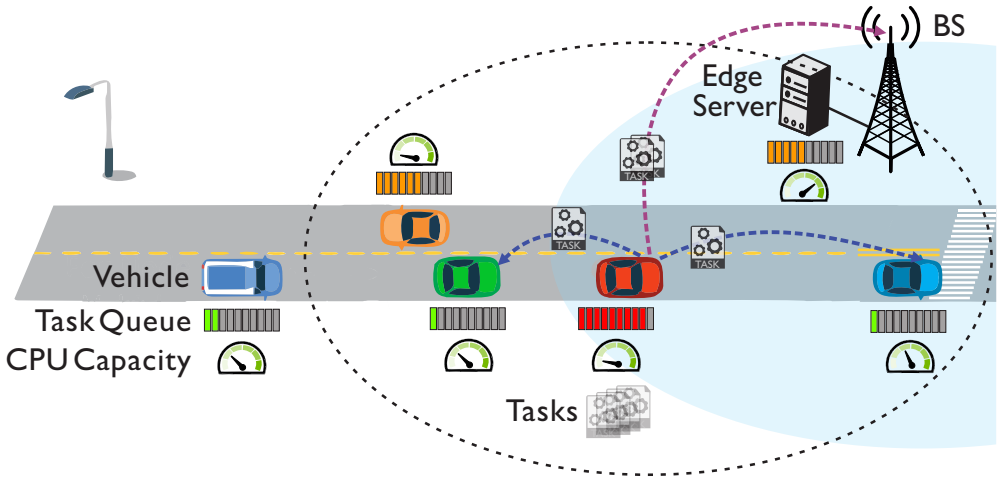
Send/Receive Tasks

After the decision, the *Decision Maker* module transfers the tasks to the *Distributor* module that forwards the tasks to the appropriate modules (for local or remote execution). If the option is to send some tasks to remote devices, they are sent to the *WAVE* or *5G* modules.

In this way, as shown in Figure 11, the client (red) distributes tasks to remote servers (vehicles and edge server) to start execution. Two tasks are sent to be executed on the edge server (whose base station is within its 5G communication range - light blue background) via V2I connection (purple dotted line). In addition, two tasks are sent to be executed by vehicles. One task goes to the vehicle ahead and another task goes to the vehicle behind (within its WAVE communication range - black dotted line) through V2V connections (blue dotted lines). Thus, multiple servers can simultaneously collaborate to provide computing services to the client.

Following processing, each chosen server returns the processing result to the client. This result comes through the *WAVE* or *5G* modules. Then, the result passes through the *Data and Context Gatherer* module and finally reaches the *Application* to continue its execution.

Figure 11 – Sending of tasks to the chosen servers.



Source: The Author.

Failure recovery

After the client sends the tasks to the remote servers, it triggers a function to monitor the connectivity between client and servers, through the *Data and Context Gatherer* module. If beacon messages from a server continue to be received by the client, connectivity still exists. If no beacon messages arrive from a server in ξ milliseconds, and if the processing result from that server has not yet been returned, a failure is detected. In this case, the *Data and Context Gatherer* module informs the *Decision Maker* module. The *Decision Maker* module activates the *Task Distributor* module that has the tasks of the lost server and distributes it to the *Task Queue* to be executed locally on the client.

5.2 Decision Algorithms

The decision process is the core of the framework described in the previous section. This process assigns each application task to execute on a server (local or remote). The *Decision Maker* module needs to have enough contextual information at its disposal to make these assignments. The contextual information considered by the *Decision Maker* module is: speed, direction, distances between devices, CPU availability, data rates and communication ranges WAVE and 5G, transmission and processing times, link lifetimes, and tasks characteristics. In addition, the last two algorithms presented in this section also consider the CPU capacity (in GHz) and special contextual information about known routes of vehicles, which helps to predict vehicle positioning more accurately and avoid offloading failures.

After ψ milliseconds of gathering this information, the module executes an algorithm to decide the assignment/scheduling of tasks[1]. The objective of the decision is to minimize the applications' execution time (problem P1), trying to optimize task distribution and following constraints C1 to C4 to avoid offloading failures and lack of energy on the network nodes. In this process, tasks can be sent to edge servers and vehicles simultaneously through 5G/V2I and WAVE/V2V connections. With this, we increase the network's transmission capacities and take advantage of all computing resources available in a VEC system.

In the first two sections that follow, two greedy algorithms that do task assignment (upload order does not matter) are presented. Section 5.2.1 presents the **G**reedy for **C**PU **F**ree (GCF) algorithm. Then, Section 5.2.2 describes the **G**reedy **T**ask by **T**ask (GTT) algorithm. Finally, an intelligent algorithm that performs task scheduling (upload order matters) called A**B**C for **C**omputation Offloading in **V**EC (BCV) is explained in Section 5.2.3.

5.2.1 Greedy for CPU Free

Greedy for CPU Free (GCF) is a decision and task assignment algorithm that prioritizes sending tasks to servers with the highest processing availability (lowest queuing time) and the shortest distances to the *client*. This prioritization is done by sorting the set of feasible servers F. Such sorting is done only once for each workload. In the event of a tie in the server evaluation, the algorithm breaks the tie by the following order of priority: edge servers, client, and nearby vehicles. After sorting, the algorithm goes through the set F and allocates as many tasks as possible to its servers.

Therefore, the GCF's pseudocode is shown in Algorithm 1 and explained below.

GCF Algorithm

According to Algorithm 1, GCF receives as input a set M of reply messages, a set T of locally generated tasks to be processed, and a variable α (number of algorithm attempts). In line 1, GCF initializes the sets S (servers) and Y (backup of T) and variable i (*client* ID). In line 2, the set F of feasible servers is initialized with a tuple from the *client*, indicating that it can also execute tasks. Afterwards, in line 3, F receives more feasible servers on return from the Function *AddFeasibleServers* (Algorithm 2).

[1] This decision can even assign all tasks for local execution. In this case, offloading is not done because it is not worth it.

Algorithm 1: GCF Algorithm

```
   Input: M, T, ɑ
   Output: result
1  S ← ∅; Y ← T; i ← getClientId();
2  F ← {i; d_{i,i}; t_{i,queue}; C_i; 0; 0; ∅};
3  AddFeasibleServers(M, F, i);
4  Sort F by (t_queue * δ + d * σ + getType());
5  foreach Z in F do
6      z ← getServerId(Z);
7      if (z = i) then
8          AddTasksToClient(i, F, T);
9      else
10         AddTasksToServer(z, F, T);
11 if (T = ∅) then
12     Process(W); SendWorkloads(S);
13     return true;
14 else
15     if (ɑ < φ) then
16         WaitForMoreReplies(ψ/5);
17         T ← Y; ɑ ← ɑ + 1;
18         GCFAlgorithm(M, T, ɑ);
19     else
20         Process(Y);
21         return false;
```

Then, in line 4, the set F of feasible servers is sorted in increasing order according to a weighted sum of values. The values considered in this sum are: queue time, distance to i, server type value ($getType()$), and the constants δ and σ. Thus, the first elements of F are the nodes with the smallest values of this weighted sum. Typically, these nodes have the lowest queuing times or shortest distances to i. As a tiebreaker in sorting, $getType()$ returns 1 (if edge server - most prioritized node), 2 (if client), or 3 (if nearby vehicle - least prioritized node). Then, in lines 5-12, the sorted set F is traversed so that its servers execute the tasks of T. In lines 7-10, if the server z is the client itself, the GCF calls the Function *AddTasksToClient* (Algorithm 3). Otherwise, the GCF calls the Function *AddTasksToServer* (Algorithm 4).

When leaving the loop, in lines 11-13, the algorithm checks if there are still tasks in T. If T is empty, the *client* processes locally a set W of tasks that belonged to T and sends the other tasks that were assigned to servers in S. Lines 14-21 treat the cases where tasks remain in T. In lines 15-18, if $\mathfrak{a}$ is less than the maximum number of attempts (φ), the GCF waits $\frac{\psi}{5}$

milliseconds for more replies, resets T, increments $\mathfrak{a}$, and calls the GCF to run again. If the number of attempts reaches the limit (lines 19-21), the *client* executes the entire initial set of tasks (Y) locally.

Function *AddFeasibleServers*

As described in Algorithm 2, this Function checks each reply message m received from a possible server j. This check is made with estimates based on Section 4.1. The purpose of the check is to know if j is feasible to perform tasks for the *client*. In line 3 of the Algorithm 2, t_j receives the estimated time to execute a task $\tau \in T$ on the possible server j. When checking lines 4-6 (partial constraint C2 of the problem P1), if t_j is less than the estimated link lifetime between i and j, F receives the tuple of the possible server j. This tuple contains, among other things, how much it will process (h_j), how much data it will receive and return (u_z), and the set of tasks that it will process (X_j).

Algorithm 2: Function *AddFeasibleServers*

```
   Input: M, F, i
1  foreach m ∈ M do
2      j ← getServerId(m);
3      t_j ← calcServerTime(τ, r_{i,j}, d_{i,j});
4      if (t_j < t_{link_{i,j}}) then
5          h_j ← 0; u_j ← 0; X_j ← ∅;
6          F ← F ∪ {j; d_{i,j}; t_{j,queue}; C_j; h_j; u_j; X_j};
```

Function *AddTasksToClient*

As can be seen in Algorithm 3, in lines 1-7, the workload T is traversed so that its tasks are assigned to the *client*. In line 2, how much the client will process (h_i) receives an addition of c_τ. In lines 3-7, the algorithm checks whether h_i exceeds the maximum processing limit per workload defined for the client. This limit is the ceiling on the maximum amount of processing for the original workload ($Y_{procmax}$) divided by a constant defined for the client (ζ_{client}). If h_i does not exceed the limit, τ is placed in the set of tasks that will be executed locally by the *client* (W) and τ is removed from T. Otherwise, c_τ is subtracted from h_i.

Algorithm 3: Function *AddTasksToClient* of the GCF

Input: i, F, T
1 **foreach** $\tau \in T$ **do**
2 $h_i \leftarrow h_i + c_\tau$;
3 **if** $(h_i < \left\lceil \frac{Y_{procmax}}{\zeta_{client}} \right\rceil)$ **then**
4 $W \leftarrow W \cup \tau$;
5 $T \leftarrow T - \tau$;
6 **else**
7 $h_i \leftarrow h_i - c_\tau$;

Function *AddTasksToServer*

As evidenced in Algorithm 4, this function assigns tasks to edge servers or nearby vehicles. In line 1, the amount of processing allocated to server z (h_z) receives the addition of c_τ, the amount of data that server z will receive (u_z) receives the addition of the upload ($s_{\tau,up}$) and download ($s_{\tau,down}$) packet size of the task τ. In line 2, the time estimate for the server z to execute its tasks is placed in the variable t_z.

Algorithm 4: Function *AddTasksToServer* of the GCF

Input: z, F, T
1 **foreach** $\tau \in T$ **do**
2 $h_z \leftarrow h_z + c_\tau$; $u_z \leftarrow u_z + s_{\tau,up} + s_{\tau,down}$;
3 $t_z \leftarrow calcServerTime(h_z, u_z, r_{i,z}, d_{i,z})$;
4 $E_{z,proc} \leftarrow \frac{h_z}{C_z} P_{z,proc}$;
5 **if** $(getType(z) = 1)$ **then**
6 $\zeta_{server} \leftarrow \zeta_{edge}$;
7 **else**
8 $\zeta_{server} \leftarrow \zeta_{vehicle}$;
9 **if** $(\ (t_z < t_{link_{i,z}})\ and\ (E_{z,cur} - E_{z,proc} > A_z)\ and\ (h_z < \left\lceil \frac{Y_{procmax}}{\zeta_{server}} \right\rceil)\ )$ **then**
10 **if** $(z \notin S)$ **then**
11 $S \leftarrow S \cup z$;
12 $X_z \leftarrow X_z \cup \tau$;
13 $T \leftarrow T - \tau$;
14 **else**
15 $h_z \leftarrow h_z - c_\tau$; $u_z \leftarrow u_z - s_{\tau,up} - s_{\tau,down}$;

Based on Section 4.1.4, in line 4 of the Algorithm 4, the estimate of energy consump-

tion when processing h_z on server z is placed in $E_{z,proc}$. In lines 5-8, if the server z is type 1, then the constant that defines the processing limit for the server (ζ_{server}) receives the defined value for the edge server (ζ_{edge}). If not, ζ_{server} receives the defined value for a server vehicle ($\zeta_{vehicle}$).

In lines 9-15, the algorithm checks whether the server z can execute its tasks (constraints C2 and C4 of the problem P1). In line 9, the algorithm first checks whether the link lifetime between z and i is sufficient to z execute its tasks without losing connectivity with the $client$. The second check evaluates whether the energy of the server z is sufficient for it to execute its tasks. The third check calculates whether h_z does not exceed the processing limit set for the server. If all conditions are met, in lines 10-11, the server z is added to the server set S, if it is not already in S. Then, in lines 12-13, the set of tasks that the server z will execute (X_z) receives τ and τ is removed from T. If line 9 conditions are not met, in lines 14-15, c_τ is subtracted from h_z and $s_{\tau,up}$ and $s_{\tau,down}$ are subtracted from u_z.

5.2.2 Greedy Task by Task

Greedy Task by Task (GTT) is a decision and task assignment algorithm that seeks the best possible server to execute each application task. The algorithm classifies a server as better mainly by the following context parameters: CPU capacity, CPU availability, and distance to the $client$. Thus, the best server tends to have high CPU capacity, low queue time, and a short distance to the $client$. Moreover, the algorithm updates the best servers list in real-time as it decides where to send tasks and uses information about known routes of vehicles. Therefore, the main characteristics of the GTT that differentiate it from the GCF are: use of contextual information of known routes of vehicles and CPU capacity; case-by-case analysis to allocate each task (first traversing the set of tasks and not the set of feasible servers); update of the best server at each allocation with different criteria.

We present the GTT's pseudocode in Algorithm 5, and we describe it below.

GTT Algorithm

After receiving the inputs, Algorithm 5 makes the initializations in lines 1 and 2. Next, the Function *AddFeasibleServers* (Algorithm 2) is called to put the feasible servers in F. Then, in line 4, the set F of feasible servers is sorted in increasing order according to the distance to i, computational capacity, queue time, and the constants σ, ρ, and δ.

In loop of lines 5-10, T is traversed task by task so that each task τ is assigned

Algorithm 5: GTT Algorithm

Input: M, T
Output: *result*

```
1  S ← ∅; Y ← T; i ← getClientId();
2  F ← {i; d_{i,i}; t_{i,queue}; C_i; 0; 0; ∅};
3  AddFeasibleServers(M, F, i);
4  Sort F by (d * σ + 1/C * ρ + t_queue * δ);
5  foreach τ ∈ T do
6      z ← getIdOfFirstElement(F);
7      if (z = i) then
8          AddTasksToClient(i, τ, F, T);
9      else
10         AddTasksToServer(z, τ, F, T);
11 if (T = ∅) then
12     Process(W); SendWorkloads(S);
13     return true;
14 else
15     Process(Y);
16     return false;
```

to a server (local or remote). The evaluation is always done by the first element of the sorted set F, that is, the best-evaluated server at the moment. In lines 7-10, if the first element is the client itself, the Function *AddTasksToClient* is called (Algorithm 6). Otherwise, the Function *AddTasksToServer* is called (Algorithm 7).

Finally, in lines 11-16, if there are no tasks left in T, the *client* processes tasks in W locally, and the other tasks are sent to be executed in servers in S. If tasks remain, a problem has occurred, and the initial set of tasks (Y) is all executed locally.

Function *AddTasksToClient*

Function *AddTasksToClient* is responsible for assigning tasks to the *client*. In Algorithm 6, W (task set to be executed locally) receives τ, that is removed from T, the queue of i is increased by the time to process τ (as if τ was already queued in i), and F is sorted again.

Function *AddTasksToServer*

Function *AddTasksToServer* manages the task assignment to remote servers. In Algorithm 7, line 1, as the server z will process the task τ, how much it will process (h_z) receives

Algorithm 6: Function *AddTasksToClient* of the GTT

Input: i, τ, F, T

1 $W \leftarrow W \cup \tau$;
2 $T \leftarrow T - \tau$;
3 $t_{i,queue} \leftarrow t_{i,queue} + \frac{c_\tau}{C_i}$;
4 Sort F by $(d * \sigma + \frac{1}{C} * \rho + t_{queue} * \delta)$;

an addition of c_τ. How much data on the server z will be received and returned (u_z) receives an addition of $s_{\tau,up}$ and $s_{\tau,down}$. In line 2, t_z receives the estimated time for server z to execute its tasks. In line 3, according to Section 4.1.4, $E_{z,proc}$ receives the estimated energy consumption when processing h_z on the server z. In lines 4-5, if the route of the server is known (K_z), l receives true if i and z are still within the communication range of each other after t_z seconds (verification by Function *withinRange* according to position prediction based on known routes of vehicles of Section 4.1.2).

Algorithm 7: Function *AddTasksToServer* of the GTT

Input: z, τ, F, T

1 $h_z \leftarrow h_z + c_\tau$; $u_z \leftarrow u_z + s_{\tau,up} + s_{\tau,down}$;
2 $t_z \leftarrow calcServerTime(h_z, u_z, r_{i,z}, d_{i,z})$;
3 $E_{z,proc} \leftarrow \frac{h_z}{C_z} P_{z,proc}$;
4 **if** $(K_z = true)$ **then**
5 $l \leftarrow withinRange(i, z, t_z)$;
6 **if** ($((K_z = true \text{ and } l = true) \text{ or } (t_z < t_{link_{i,z}}))$ *and* $(E_{z,cur} - E_{z,proc} > A_z)$) **then**
7 **if** $(z \notin S)$ **then**
8 $S \leftarrow S \cup z$;
9 $X_z \leftarrow X_z \cup \tau$;
10 $T \leftarrow T - \tau$;
11 $t_{z,queue} \leftarrow t_{z,queue} + \frac{c_\tau}{C_z}$;
12 Sort F by $(d * \sigma + \frac{1}{C} * \rho + t_{queue} * \delta)$;
13 **else**
14 $h_z \leftarrow h_z - c_\tau$; $u_z \leftarrow u_z - s_{\tau,up} - s_{\tau,down}$;
15 $t_{z,queue} \leftarrow t_{z,queue} + \varsigma$;
16 Sort F by $(d * \sigma + \frac{1}{C} * \rho + t_{queue} * \delta)$;

In lines 6-16, the algorithm checks whether the server can execute its assigned tasks (constraints C2 and C4 of the problem P1). With the estimate of the execution time of the

assigned tasks, the first verification identifies whether connectivity will exist until the end of the execution of h_z on the server z. This check is done by predicting positioning, if the route is known, or by the estimated link lifetime. After the second "and" keyword, the second verification detects if the server z has enough energy to execute h_z. Suppose the verifications return $true$ (lines 6-12). Then, if the server is not in S, it is added. Next, X_z receives τ, τ is removed from T, the task queue of the server is increased by the time to process τ (as if the task τ was already queued at *server*), and F is sorted again. Suppose the check returns $false$ (lines 13-16). In that case, the procedures for executing τ are undone, the queue of the server receives a high value constant ς so that the server stays in the last positions of F, and F is sorted again.

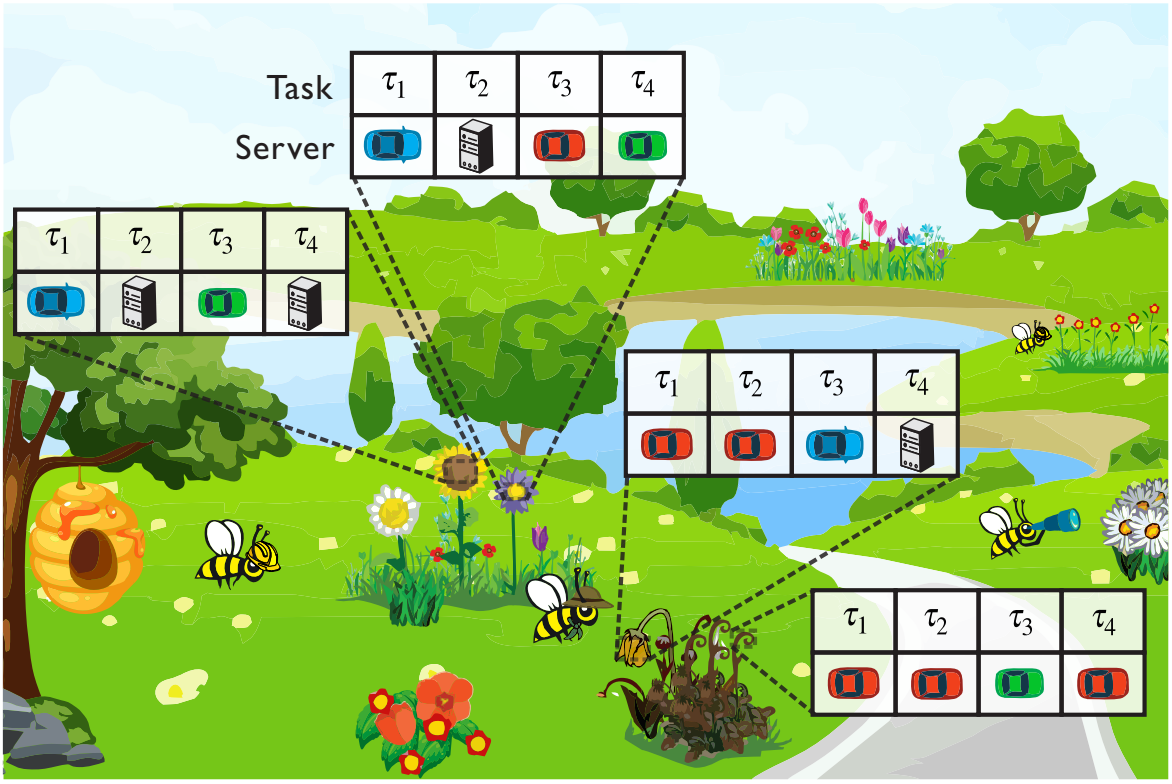
5.2.3 ABC for Computation Offloading in VEC

A**B**C for **C**omputation Offloading in **V**EC (BCV) is a decision, task scheduling, and intelligent algorithm based on the ABC metaheuristic. This algorithm is inspired by the behavior of honey bees when searching for food sources. As shown in Figure 12, the BCV considers a food source (nectar from a flower) as a feasible solution to problem P1. This solution is an association between each task in the workload (τ_1, τ_2, τ_3, τ_4) and the server where it will be executed. An infeasible solution is one that can cause offloading failures. These possible failures are predicted by calculations of the algorithm. They indicate a future lack of connectivity between *client* and *server* or of energy of some node.

Figure 12 uses the scenario in Figure 10, which shows the servers available for the *client* (red vehicle). The algorithm evaluates the fitness of each solution based on t_{total} (the sum of the time to execute each workload task on the different chosen servers). The lower t_{total}, the greater the fitness of the solution. In Figure 12, we can see that the search space in the region of withered flowers is related to more task executions in the *client* itself. The poor quality of the flowers (low fitness) is because, according to Figure 10, the *client* has a low CPU capacity and its queuing time is high. Thus, the t_{total} tends to be higher, decreasing the fitness of the solutions in this region of the search space. The orange vehicle does not appear in the tables of the solutions because it would make any solution infeasible. As it is in the opposite direction of the "client", it will quickly lose connectivity and will not return the result of processing a workload task.

The BCV's pseudocode is shown in Algorithm 8. We also explain the algorithm in the following paragraphs.

Figure 12 – Figurative depiction of bees looking for solutions in the BCV search space.



Source: The Author.

BCV Algorithm

Algorithm 8: BCV Algorithm

Input: M, T
Output: *result*

$S \leftarrow \emptyset$; $V \leftarrow \emptyset$; $i \leftarrow getClientId()$; $y_{cycles} \leftarrow 0$;
$F \leftarrow \{i; d_{i,i}; t_{i,queue}; C_i; 0; 0; \emptyset\}$;
$AddFeasibleServers(M, F, i)$;
$InitializeFoodSources(F, T, V, i)$;
while $(y_{cycles} < \eta_{cycles})$ **do**
    $EmployedBees(F, T, V, i)$;
    $OnlookerBees(F, T, V, i)$;
    $ScoutBees(F, T, V, i)$;
    $H \leftarrow BestFoodSource(V)$;
    $UpdateCyclesOfFoods(V)$;
    $y_{cycles} \leftarrow y_{cycles} + 1$;
$W \leftarrow ExtractTasksToLocal(H)$; $S \leftarrow ExtractRemoteServersAndTheirTasks(H)$;
$Process(W)$;
$ScheduleUpload(H)$; $SendWorkloads(S)$;
return *true*;

The BCV algorithm initializes the sets of servers (S) and victuals/foods sources (V) as empty and initializes the identification of the *client* i and the cycles counter (y_{cycles}) in line 1. Subsequently, in lines 2 and 3, BCV initializes the set F of feasible servers with the parameters of the *client* itself and calls the Function *AddFeasibleServers* (Algorithm 2) to put feasible remote servers in F.

In line 4, BCV calls the Function *InitializeFoodSources* (Algorithm 9) to do global searches for food sources, added to the set V. In global searches, all servers are chosen at random to be associated with a task. In lines 5-11, the loop is executed for a predetermined number of cycles (η_{cycles}). In line 6, the algorithm calls the Function *EmployedBees* (Algorithm 10) for employed bees to do local searches around existing food sources. The local search tries to find feasible solutions within the same region of the search space. According to Figure 12, all solutions in the same region of the search space (heap of nearby flowers) have the same first half of the task/server table.

In line 7, the onlooker bees also do local searches around the best food sources, according to the Function *OnlookerBees* (Algorithm 11). In line 8, the Function *ScoutBees* (Algorithm 12) is called so that, if a food source is abandoned, scout bees do global searches to find and evaluate a new food source. Then, in line 9, the best food source, with the highest fitness value, is chosen and placed in H. In line 10, the duration counter for each food source in the set V is incremented. This counter is related to the number of cycles that a food source remains in V. The cycles counter is also incremented in line 11.

Finally, in line 12, the algorithm already has the food source with the highest fitness (H) among all those evaluated in V over the cycles. The solution/food source contains the best way to distribute the tasks to the servers, among all the evaluated solutions. According to it, W receives the tasks that will be processed locally and S receives the remote servers and the tasks that they will process. Thus, in line 13, the *client* processes its workload W. In line 14, the algorithm organizes the order of uploading tasks to their respective servers, prioritizing immediate sending to the edge server and then to server vehicles with the lowest link lifetime with i. Then, the workloads are sent to their servers. In line 15, the algorithm returns $true$, indicating that the task distribution decision was successful.

Function *InitializeFoodSources*

The Function *InitializeFoodSources* (Algorithm 9) is responsible for placing η_{foods} food sources in the set V, which is initially empty. Thus, after initializing the food sources counter (y_{foods}, line 1), the loop, in lines 2-7, for filling the set V begins. In line 3, the solution or grub/food source being prepared (G) is initialized. In line 4, the algorithm calls the Function *Search* (Algorithm 13). Since G is empty, this call does a global search and finds a feasible food source, assigning it to G. In line 5, the fitness f of G is calculated. This calculation is based on context information and estimates the total offloading time (see Section 13). Thus, the lower the total offloading time estimate for the current solution G, the greater its fitness. Then, in line 6, V receives a tuple with the current solution G, its fitness (f), and the duration in cycles of G in V (initially zero).

Algorithm 9: Function *InitializeFoodSources* of the BCV

```
   Input: F,T,V,i
1  y_foods ← 0;
2  while (y_foods < η_foods) do
3      G ← ∅;
4      Search(F,T,∅,G,i);
5      f ← getFitness(G);
6      V ← V ∪ {G,f,0};
7      y_foods ← y_foods + 1;
```

Function *EmployedBees*

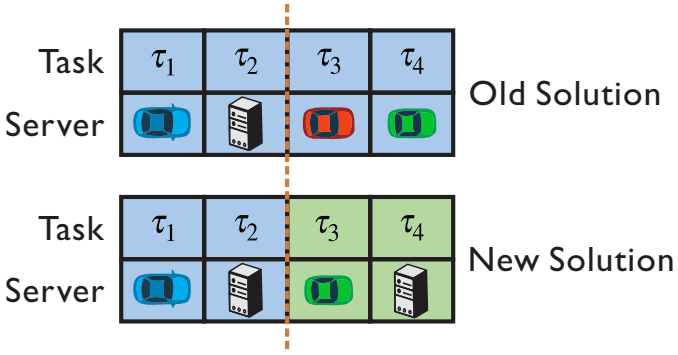
In the loop of lines 1-4, the Function *EmployedBees* (Algorithm 10) checks for neighboring solutions of each existing solution in V. In line 2, the current solution G being prepared receives the first half of an existing solution I. In line 3, as G has only half of the tasks of T allocated, the set U receives the unallocated tasks from T. In line 4, using the Function *Search* (Algorithm 13), the algorithm does a local search to assign the tasks of U to servers, complete the solution G, and evaluates whether it is worth changing I for G based on their fitness.

Figure 13 shows how the local search for employed and onlooker bees is carried out. We can see that the first half of the old solution (blue background) remains in the new solution. The second half of the new solution (green background) receives servers chosen randomly.

Algorithm 10: Function *EmployedBees* of the BCV

```
   Input: F, T, V, i
1  foreach I in V do
2      G ← getFirstHalf(I);
3      U ← getUnallocatedTasks(T, G);
4      Search(F, U, I, G, i);
```

Figure 13 – A local search using the BCV algorithm.



Source: The Author.

Function *OnlookerBees*

This function (Algorithm 11) intensifies the search for food sources in the vicinity of the best food sources already identified through the loop of lines 2-15 (controlled by the onlooker bees counter ($y_{onlookers}$)). In lines 3-11, the algorithm performs a tournament to choose, among $\eta_{tournament}$ food sources randomly selected from V, the best one. In lines 3 and 4, H is assumed to be the best food source, when it is also randomly selected from V and its fitness f is calculated. In the loop of lines 6-11, the tournament is held, and H receives the best food source, among those selected for the tournament. Then, in lines 12-14, using the Function *Search* (Algorithm 13), the algorithm searches for a food source in the vicinity of H to try to find a better solution, following the same steps as lines 2-4 of the Algorithm 10.

Function *ScoutBees*

The Function *ScoutBees* (Algorithm 12) performs global searches looking for new food sources after some of them have been abandoned. In line 1, the algorithm sorts V in decreasing order of fitness of the food sources. Thus, in line 2, V_{shalf} receives the second half of V, i.e., it receives the set of food sources with the worst fitness. In the loop of lines 4-6, the set V_{shalf} is traversed. In this loop, the function checks whether the duration of each food source exceeds a threshold ($\eta_{abandon}$), which indicates when a solution should be abandoned. Suppose

Algorithm 11: Function *OnlookerBees* of the BCV

```
   Input: F, T, V, i
1  y_onlookers ← 0;
2  while (y_onlookers <= η_onlookers) do
3      H ← getFoodSourceRandomly(V);
4      f ← getFitness(H);
5      y_tournament ← 0;
6      while (y_tournament < η_tournament) do
7          G ← getFoodSourceRandomly(V);
8          if (getFitness(G) > f) then
9              H ← G;
10             f ← getFitness(H);
11         y_tournament ← y_tournament + 1;
12     G ← getFirstHalf(H);
13     U ← getUnallocatedTasks(T, G);
14     Search(F, U, H, G, i);
15     y_onlookers ← y_onlookers + 1;
```

the threshold is exceeded. In that case, with the Function *Search* (Algorithm 13), the algorithm globally searches for a new solution. Then it evaluates whether it is worth replacing the previous solution with the new one.

Algorithm 12: Function *ScoutBees* of the BCV

```
   Input: F, T, V, i
1  Sort V by fitness;
2  V_shalf ← getSecondHalf(V);
3  foreach I ∈ V_shalf do
4      if (getDuration(I) >= η_abandon) then
5          G ← ∅;
6          Search(F, T, I, G, i);
```

Function *Search*

This function (Algorithm 13) performs searches in order to find and assemble feasible solutions. In the loop of lines 1-6, the function seeks to assign each task τ from U to a server (local or remote). In line 2, the variable indicating whether a task/server assignment is feasible (o) is initialized with $false$. The algorithm remains in a loop in lines 3-5 until it finds a feasible task/server assignment. In line 4, a server z is chosen at random from F. In line 5, the Function *CheckFeasibility* analyzes whether it is feasible to run τ on the server z. If feasible, the algorithm

exits the loop and places the task/server pair ($\{\tau, z\}$) in the current solution G. Thus, G is being assembled so that all tasks of U are linked to servers, representing a way of distributing tasks and doing computation offloading.

Algorithm 13: Function *Search* of the BCV

```
   Input: F,U,I,G,i
1  foreach τ ∈ U do
2      o ← false;
3      while (o = false) do
4          z ← getServerRandomly(F);
5          o ← CheckFeasibility(G,τ,z,i);
6      G ← G ∪ {τ,z};
7  if (I ≠ ∅) then
8      f ← CalculateFitness(G);
9      if (f > getFitness(I)) then
10         I ← {G,f,0};
```

Then, in lines 7-10, it is checked whether the new solution needs to be compared with any previous solution. If I is empty, it indicates that there is no previous solution (in case it happens in Function *InitializeFoodSources*). If there is a previous solution (in the case of Functions *EmployedBees*, *OnlookerBees*, and *ScoutBees*), the fitness f of the current solution G is calculated (line 8). Suppose it is greater than the fitness of the previous solution (lines 9-10). In that case, the previous solution is replaced by the current solution.

Function *CheckFeasibility*

The feasibility check for assigning a task τ to a server z is done by the Function *CheckFeasibility* (Algorithm 14). In lines 1-3, if the server z is the *client* i itself, the task τ is assigned to the set of tasks that will be executed locally by the *client* (W_G) in this solution G (line 2). Since this assignment is feasible, the function's return is $true$. If z is not the *client* (lines 4-15), it is necessary to make a feasibility assessment. In the solution G, how much z will process (h_{G_z}) receives the addition of c_τ (line 5). Then, how much data z will receive in the solution G (u_{G_z}) receives the amount of upload and download data for the task τ. In line 6, the estimated time that z will take to execute the task τ in solution G (t_{G_z}) is calculated. In line 7, the energy consumption estimate is calculated for z to process its tasks in solution G ($E_{z,proc}$).

Later, lines 8-9 check if the route of z is known (K_z). If so, l receives true if i and z

Algorithm 14: Function *CheckFeasibility* of the BCV

Input: G, τ, z, i

1 **if** $(z = i)$ **then**
2 $W_G \leftarrow W_G \cup \tau$;
3 **return** $true$;
4 **else**
5 $h_{G_z} \leftarrow h_{G_z} + c_\tau$; $u_{G_z} \leftarrow u_{G_z} + s_{\tau,up} + s_{\tau,down}$;
6 $t_{G_z} \leftarrow calcServerTime(h_{G_z}, u_{G_z}, r_{i,z}, d_{i,z})$;
7 $E_{z,proc} \leftarrow \frac{h_{G_z}}{C_z} P_{z,proc}$;
8 **if** $(K_z = true)$ **then**
9 $l \leftarrow withinRange(i, z, t_{G_z})$;
10 **if** $(\ ((K_z = true \ and \ l = true) \ or \ (t_{G_z} < t_{link_{i,z}}))\ and\ (E_{z,cur} - E_{z,proc} > A_z)\ )$ **then**
11 $X_{G_z} \leftarrow X_{G_z} \cup \tau$;
12 **return** $true$;
13 **else**
14 $h_{G_z} \leftarrow h_{G_z} - c_\tau$; $u_{G_z} \leftarrow u_{G_z} - s_{\tau,up} - s_{\tau,down}$;
15 **return** $false$;

will be still within the communication range of each other after t_{G_z} seconds (see Section 4.1.2). In lines 10-15, the algorithm checks whether it is feasible for the server z to execute its tasks, including the newly added task τ. For this, the constraints C2 and C4 of problem P1 are analyzed. In line 10, if the route of the server z is known, the connectivity between it and the *client* is verified by calculating the position prediction. Suppose the z route is not known. In that case, the connectivity between z and i is evaluated by the estimated link lifetime $t_{link_{i,z}}$. Still, in line 10, the algorithm checks if z has enough energy to execute h_{G_z}. Suppose the requirements of line 10 are met. In that case, the set of tasks that z will process according to the solution G (X_{G_z}) receives τ (line 11). Next, the function returns $true$, indicating that it is feasible to execute τ on the server z (line 12). If the requirements of line 10 are not met (lines 13-15), τ is removed from what z will receive and process, and the function returns $false$, indicating that it is not feasible to execute τ on server z.

5.3 Concluding Remarks

This chapter introduced the architecture of a new context-oriented framework for computation offloading in VEC systems. The different modules of the architecture were described, as well as the flow of information between them. This framework allows WAVE and

5G communication technologies to be used to take advantage of the computing resources of the edge and vehicular clouds. The framework supports several stages of the computation offloading process, such as resources discovery, gathering various context parameters, decision and task distribution, and failure recovery.

This chapter also described the pseudocodes for the three new decision algorithms proposed. These algorithms aim to solve the problem P1 explained in Section 4.2. They try to decrease the execution time of vehicular applications by offloading computation tasks in the best possible way to different servers. In addition, they make link lifetime and energy consumption estimates and checks to satisfy the mobility and energy constraints of problem P1.

The strategy of the GCF algorithm is to distribute tasks prioritizing servers that have the lowest queue times and shortest distances to the client. The strategy of the GTT algorithm is to choose the best possible server to execute each task of the workload. The best server tends to have high CPU capacity (in GHz), low queue time, and a short distance to the client. Moreover, with each new task assignment, the algorithm updates the server's queue time (as if the task was already queued) and the classification of the best server. Finally, the BCV algorithm finds and evaluates several feasible solutions in the search space, using the solution with the best fitness found for the computation offloading process. The fitness of a solution is inversely proportional to its estimated total execution time. The BCV is inspired by the behavior of honey bees in search of food sources. Besides, both GTT and BCV use data of complete routes of vehicles (if available). In this way, the vehicle is aware of the trajectory of other vehicles to their destinations, helping to make more accurate predictions about connectivity.

6 EVALUATION

This chapter presents the detailed configurations of the experiments and preliminary tests of parameters carried out. In addition, as the main objective of this chapter, we also present the extensive experiments of the proposed solutions and literature algorithms. Thus, it possible to evaluate and validate them in different vehicular environments in terms of execution time and reliability.

The rest of this chapter is organized as follows. Section 6.1 describes the experimental setup of the simulations performed. Section 6.2 details some preliminary assessments of parameters such as the communication range, the percentage of known routes of vehicles, and the number of cycles and foods of the BCV algorithm. The most important assessment is made in Section 6.3. It presents the analyzes and discussions of the experiments of the proposed solutions and literature algorithms. In Sections 6.2 and 6.3, we show some results with averages and confidence intervals. In these results, when there are overlaps of these intervals, the performance evaluation becomes inconclusive (BELIA *et al.*, 2005; AUSTIN; HUX, 2002). Therefore, in these two sections, we also present additional statistical tests to determine whether there is a statistically significant difference in the data distributions with overlapping confidence intervals. Finally, Section 6.4 presents the final considerations of the chapter.

6.1 Experimental Setup

This section presents the details of the experiments performed to evaluate the proposed framework and algorithms. All experiments follow the general network structure, communication, computation, and energy models described in Section 4.1. Table 6 presents a summary of the main characteristics of the experiments.

Aspects of network, scenario, mobility, vehicular density, application, computation, energy, and algorithms are detailed below.

Network

We employ simulation-based experiments to evaluate the proposed framework. For this, we used the ns-3 simulator (RILEY; HENDERSON, 2010) (version 3.29) with an additional 5G/mmWave module attached (MEZZAVILLA *et al.*, 2018). The simulations were performed on a computer with an Intel Xeon E5645 processor @ 2.40 GHz and 32 GB RAM.

Table 6 – Main simulation parameters.

General	
Scenarios	Highway and Urban
# of vehicles in highway	11, 55, and 120 per km
# of vehicles in urban	25, 120, and 250 per km^2
# of vehicles off in urban	$\approx$ 10 % of total vehicles
# of CVs and EVs	50 and 50 %
Simulation time	150 seconds
# of simulations carried out	200 times
Mobility model	Krauss
Servers offering offloading	All (vehicles and edge servers)
Vehicles with known routes	50 %
CPU Capacity of Edge Servers	[1.5, 2.5] GHz
CPU Capacity of Vehicles	[0.5, 1.0] GHz
CPU Requirement of a Task	[3.5, 6.5] Gigacycles
Task size (upload)	[350, 650] kB
Result size (download)	1 kB
Workload type	ALPR
Number of tasks per workload	[4, 12] tasks
Transport protocol	UDP (discovery), TCP (offloading)
Others packets traffics	Beacon messages
WAVE	
Communication range	250 meters
Radio propagation model	Two-Ray Ground
Antenna	Omnidirectional
Layer 2 protocol	IEEE 802.11p
Data rate	27 Mbps
5G	
Communication range	220 meters
Radio propagation model	5G mmWave systems
Antenna	MIMO beamforming
Layer 2 protocol	5G mmWave systems
Data rate	450 Mbps

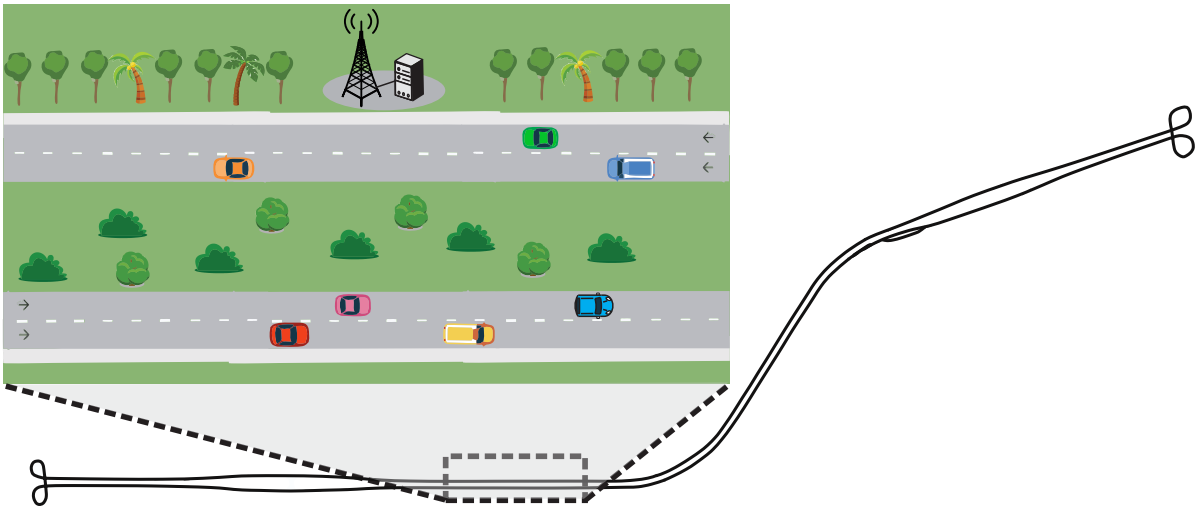
Source: The Author.

All scenarios use common packet traffic on vehicular networks, mainly periodic beacon messages. The *client* is chosen randomly among all vehicles in the scenario. As mentioned in Section 4.1.1, it is configured with WAVE and 5G interfaces and the other vehicles only with WAVE interface. The time that the *client* starts the computation offloading process is also chosen at random.

Scenario

To build the simulated scenarios, we used Simulation of Urban MObility (SUMO) (KRAJZEWICZ, 2010). The first scenario, seen in Figure 14, consists of an adapted stretch of a Brazilian highway with the following characteristics: 5 km long, two lanes in each direction, returns at the ends, and a maximum speed of 60 km/h (default limit of the chosen stretch). The black dotted lines highlight a smaller section of the highway.

Figure 14 – Highway scenario used in the experiments.



Source: The Author.

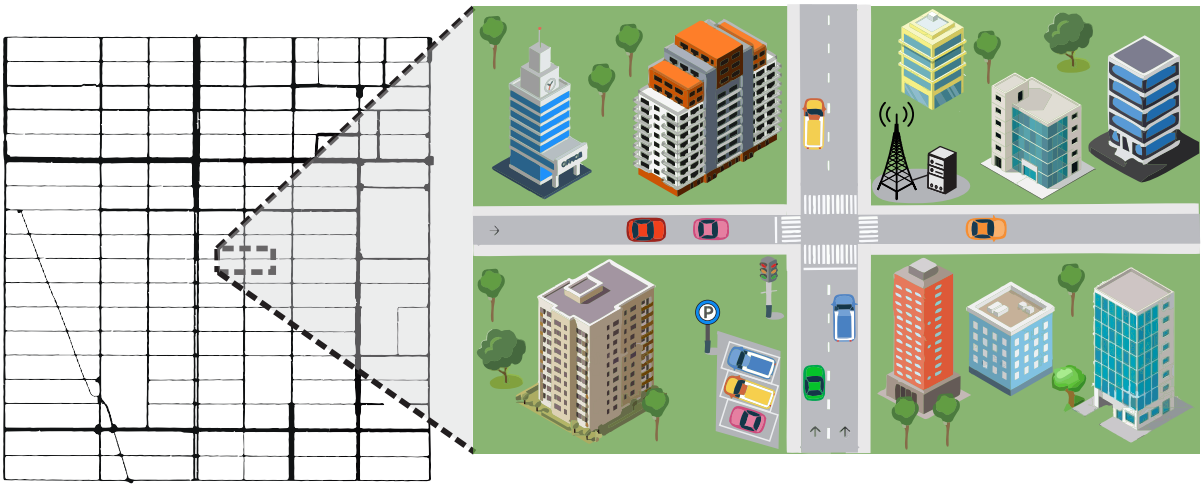
The second scenario, seen in Figure 15, consists of an adapted urban stretch of the Manhattan region, New York, USA, with 2 km^2 area and a maximum speed of 60 km/h. The black dotted lines highlight an intersection in a smaller section of the urban area. Both areas highlighted in Figures 14 and 15 show vehicles and 5G base stations with coupled edge servers.

Mobility

We use the SUMO to generate the mobility of the network nodes. These nodes are of three types with predefined quantity: vehicles, edge servers, and base stations. Base stations and edge servers have a fixed location and are spatially distributed to provide complete communication coverage.

Concerning vehicles, their initial positions are spatially and randomly distributed across different points in the scenario. They follow random paths with different starting and ending points. These paths are defined by sets of points recorded in a trace file (also used in

Figure 15 – Urban scenario used in the experiments.



Source: The Author.

disclosing known routes of vehicles). The vehicles move at different speeds and directions, according to the microscopic car-following model of Krauss. Thus, the vehicle's speed depends on the maximum speed of the road, the speed of the vehicle ahead (if any and not to collide), the difference in vehicle positions, and static parameters such as the driver's reaction time (SONG *et al.*, 2014; KRAJZEWICZ, 2010).

Vehicular Density

For each scenario, we use three types of vehicular density: low, medium, and high. Vehicular density is considered low (D_{low}) if it has approximately 11 vehicles/km in highway scenario and 25 vehicles/km^2 in urban scenario. In medium density (D_{medium}) there are approximately 55 vehicles/km in highway scenario and 120 vehicles/km^2 in urban scenario. Finally, in high density (D_{high}), we have approximately 120 vehicles/km in highway scenario and 250 vehicles/km^2 in urban scenario (SOUZA *et al.*, 2020).

Application

The application used was Automatic License Plate Recognition (ALPR), which consists of image capture, vehicle detection, plate detection, and optical character recognition. For each captured image, we used SSD-300 with MobileNet (HOWARD *et al.*, 2017) to detect vehicles and Tiny YOLOv3 (REDMON; FARHADI, 2018) to identify the license plate. Then, we used an algorithm of optical character recognition to recognize the characters. As this application's tasks are compute-intensive, the vehicles can offload them to be executed on remote

servers (edge servers or other vehicles) to reduce the execution time.

Computation

In our experiments, we considered ALPR tasks involving independent images. The result of processing each of these tasks is a string containing each recognized license plate. The packet size with the result of the processing ($s_{\tau,down}$) for any task is 1 kB.

Table 7 presents the main specifications for ALPR workloads. 10 workloads were considered ($T_1, T_2, ..., T_{10}$). Each workload has its number of tasks varying between 4 and 12. The first part of the experiments took into account only the workloads T_1 to T_5. All tasks in these workloads have the same data size for upload ($s_{\tau,up}$), in kB, and the same requirement for CPU cycles (c_τ), in Gigacycles (Gc). The second part of the experiments considered only the workloads T_6 to T_{10}, whose tasks have different values for $s_{\tau,up}$ and c_τ (CHEN *et al.*, 2020; ZHANG *et al.*, 2019).

Table 7 – ALPR workloads specifications.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
# of tasks	4	6	8	10	12	4	6	8	10	12
$s_{\tau,up}$ (kB)	558	558	558	558	558	[350, 650]	[350, 650]	[350, 650]	[350, 650]	[350, 650]
c_τ (Gc)	3.5	3.5	3.5	3.5	3.5	[3.5, 6.5]	[3.5, 6.5]	[3.5, 6.5]	[3.5, 6.5]	[3.5, 6.5]

Source: The Author.

We used real experiments for workload processing, calculating the execution time and the packet size of ALPR tasks. CPUs with capacities of 1.5, 2.0, and 2.5 GHz were used to represent edge servers. To represent vehicles, we used CPUs with 0.5 and 1.0 GHz (CHEN *et al.*, 2020; ZHANG *et al.*, 2019; MIDYA *et al.*, 2018). Using the execution time and packet size previously calculated, we simulated the computation offloading process in ns-3.

Energy Consumption

Half of the vehicles used in the simulations were EVs, and the other half were CVs. Among these, in simulations with the urban scenario, about 10 % were parked and turned off[1]. EVs require a minimum energy level (A) of 4 KWh to participate in the computation offloading processes. The stored energy levels (E_{cur}) of the EVs used in the simulation were between 5 to 20 KWh (BATTERY UNIVERSITY GROUP, 2020; HOLDING, 2020; GROUPE

[1] We did not place vehicles parked and turned off in simulations of the highway scenario.

RENAULT, 2020). CVs can be divided into two categories regarding energy: CVOs and CVMs. As the electronic part of CVOs remains active, to be part of a vehicular cloud, they also need to guarantee a minimum energy level (A), in the case of 612 Wh. The stored energy levels (E_{cur}) of the CVOs in the simulations were in the range between 620 to 720 Wh (REIS *et al.*, 2017; BATTERY UNIVERSITY GROUP, 2019; LAUKKONEN, 2019). In addition, we consider that edge servers and CVMs do not need a minimum energy level (see Section 4.1.4). Thus, the simulations did not take into account their levels of stored energy.

Moreover, according to Section 4.1.4, we consider two types of energy consumption in the computation offloading processes: task transmission (upload) and processing. Task transmissions using WAVE consume 0.046 Wh ($P_{trans,}_{WAVE}$) of energy (NS-3 TEAM, 2021; MALANDRINO *et al.*, 2014; HONG *et al.*, 2009). Task transmissions using 5G consume 1 Wh ($P_{trans,}_{5G}$) of energy (POLESE, 2016). The energy consumption related to task processing depends on the specifications of the CPU chip (P_{proc}). In vehicles with CPUs with capacities between 0.5 and 1 GHz, we considered the energy consumption with processing ranging from 20 to 30 Wh. In edge servers, with CPUs from 1.5 to 2.5 GHz, the variation in energy consumption when processing tasks was 40 to 60 Wh (INTEL, 2021a; INTEL, 2021b).

Algorithms

We compared the performance of five algorithms: the three algorithms in Section 5.2 (GCF, GTT, and BCV) and two other algorithms - FIFO (First In, First Out) and HVC (Hybrid Vehicular edge Cloud) (FENG *et al.*, 2018). All algorithms use the failure recovery mechanism described in Section 5.1.2. The FIFO algorithm selects the servers that reply first, sending a task to each one. If the FIFO does not find enough servers, it executes all tasks locally. The HVC algorithm prioritizes sending the requests via 5G (in our scenario, to the edge server). Besides, these last two algorithms do not consider the contextual information of known routes of vehicles.

6.2 Preliminary Evaluation of Parameters

This section reports the impacts caused by different and important parameters in the computation offloading processes. Sections 6.2.1, 6.2.2, and 6.2.3 present the analysis of the results related to the experiments about communication ranges, the number of cycles and foods

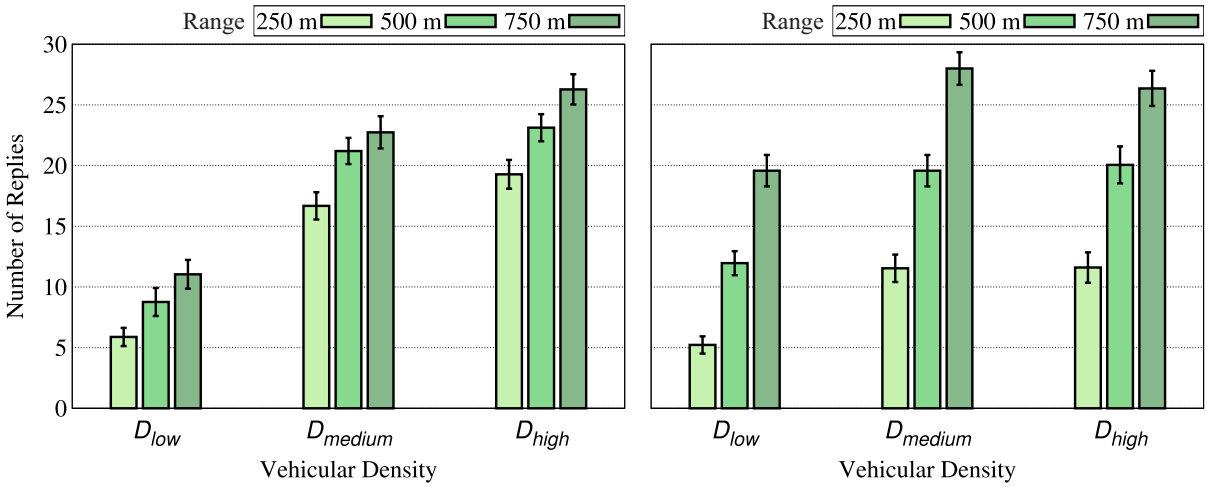
of the BCV algorithm, and known routes of vehicles. Then, in Section 6.2.4, discussions and evaluations of the results are made. We call these evaluations preliminary because they helped us to define the WAVE range, the values of BCV parameters, and the percentage of vehicles with known routes to be used in the performance evaluation of the proposed algorithms, the main experiments of this thesis (Section 6.3).

6.2.1 Communication Range

This section analyzes the impact of the WAVE communication range on highway and urban scenarios with different vehicular densities. For this, we count the number of reply messages returned to the *client* vehicle after it sends a request message, according to Section 5.1.2. In this experiment, the *client* sends a broadcast message of one-hop to request contextual information from all WAVE devices within its communication range. These devices send a reply message back to the *client*. Finally, all reply messages are counted.

In the results presented in Figure 16, 50 simulations were carried out for each range/density configuration to calculate the average number of reply messages with a 95 % confidence interval (CI). We can see in these figures that the greater the WAVE communication range, the greater the number of reply messages received by the *client*.

Figure 16 – Number of replies for different WAVE communication ranges.



(a) Highway scenario.

(b) Urban scenario.

Source: The Author.

In the case of Figure 16a, which presents data from the highway scenario, the WAVE range change from 250 to 500 m increased the average number of replies by up to 48.9 %, going from 5.8 to 8.7 with low vehicular density (D_{low}). With the range variation from 500 to 750 m,

the average number of replies increased up to 26.0 %, going from 8.7 to 11.0 with D_{low}. Figure 16b presents the data related to the urban scenario. We observe that varying the WAVE range from 250 to 500 m increased the average number of replies by up to 129.1 % (from 5.22 to 11.9 with D_{low}). By varying the range from 500 to 750 m, the maximum increase in the average amount of replies was 63.7 % (from 11.9 to 19.5 with D_{low}).

Figures 16a and 16b also show that increasing the vehicular density from D_{medium} to D_{high} does not increase the average number of replies much. This variation in vehicular density increases the average number of replies by a maximum of 15.5 % in the highway scenario (Figure 16a with 250 and 750 m). In the urban scenario (Figure 16b), the maximum increase in the average number of replies was only 2.4 % (with 500 m), even decreasing the average number of replies by 5.8 % (with 750 m).

Statistical Tests

As Figure 16 shows, there is an overlap between confidence intervals. Therefore, we need to perform further tests to determine if there is a statistically significant difference in the number of replies using different WAVE ranges. In this assessment, the ranges are tested under the same conditions. There is no sphericity in the generated data, and it is normally distributed[2]. Thus, we performed repeated measures ANOVA (rANOVA) and Tukey tests for the evaluation (SHESKIN, 2000).

We used urban and highway scenarios with three vehicular density levels, resulting in six groups of experiments. Each rANOVA test compared the number of replies for the three ranges in each group of experiments. Then, we used rANOVA to test the null hypothesis ($H0$: the average number of replies is the same for all three ranges). In all rANOVA tests, we got a $p-value < 0.05$, which rejected the null hypothesis.

Once $H0$ of the rANOVA test is rejected, we applied the Tukey method to determine which samples' pairs have statistically significant differences between their averages. Tukey test compares two-by-two samples on the left side. The null hypothesis ($H0$) of this test is that the corresponding averages are equal. However, as seen in Table 8, in only one of the cases, $H0$ is not rejected. This happens when comparing the 500 and 750 m ranges in the highway scenario with medium density (D_{medium}), where the $p-value > 0.05$. With this scenario, the ranges of 500 and 750 m show statistically the same performance in terms of number of replies.

[2] The Shapiro-Wilk test assessed normality. Mauchly's test assessed sphericity (VERMA, 2015).

Table 8 – Tukey tests p-values for WAVE ranges.

Ranges	Highway			Urban		
	$\mathbf{D_{low}}$	$\mathbf{D_{medium}}$	$\mathbf{D_{high}}$	$\mathbf{D_{low}}$	$\mathbf{D_{Medium}}$	$\mathbf{D_{high}}$
250 vs. 500 m	0.000	0.000	0.000	0.000	0.000	0.000
250 vs. 750 m	0.000	0.000	0.000	0.000	0.000	0.000
500 vs. 750 m	0.000	0.088	0.000	0.000	0.000	0.000

Source: The Author.

6.2.2 BCV Algorithm Parameters

As the BCV algorithm is based on the ABC metaheuristic, it has parameters whose values can be adjusted to a specific problem. Two important BCV parameters that require adjustments and special attention are the number of cycles (η_{cycles}) and the number of foods (η_{foods}). The number of cycles serves as a stopping criterion for the algorithm. The number of foods determines the maximum size of the solution set. Thus, this experiment evaluates the influence of these two parameters of the BCV algorithm on the computation offloading processes. We use scenarios with high vehicular density because they offer more solution options (more servers available). The CPU used in the execution of the experiments was 1 GHz and represented the CPU of a *client* vehicle of the simulations. We also use workloads with the largest number of tasks (12), taking more time to allocate them and making it difficult to build feasible solutions.

In this experiment, each combination of $\eta_{cycles}/\eta_{foods}$ was repeated 50 times to calculate two metrics with a 95 % CI. The first metric is the average time that the BCV algorithm takes to find the final solution (algorithm time). The second metric is the average reduction in the workload execution time. This last metric counts the averages of the total execution times of the solutions chosen by BCV. Then these last averages are compared with the benchmark. In turn, the benchmark is composed of the averages of the execution times when the *client* processes all tasks locally.

Tables 9 and 10 show the results of this experiment for highway and urban scenarios, respectively. According to these tables, only three combinations of $\eta_{cycles}/\eta_{foods}$ have an average algorithm time below 0.1 s: 20/20, 20/50, and 50/20. In addition, for the different combinations of $\eta_{cycles}/\eta_{foods}$, there are averages with different values for the "reduction in execution time" metric. However, the confidence intervals of these averages overlapped. Thus, statistical tests that were carried out to verify that these averages are statistically equal are presented below.

Table 9 – Time values by the BCV algorithm cycles and foods in the highway scenario.

$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$	**Algorithm Time (s)**	**CI**	**Reduction in Execution Time (%)**	**CI**
20	20	0.028	± 0.007	66.440	± 4.405
20	50	0.063	± 0.008	66.918	± 4.359
20	100	0.143	± 0.009	64.819	± 5.239
50	20	0.065	± 0.008	66.242	± 4.414
50	50	0.179	± 0.010	67.074	± 4.044
50	100	0.364	± 0.017	64.109	± 5.886
100	20	0.147	± 0.010	66.323	± 4.996
100	50	0.374	± 0.016	66.903	± 4.702
100	100	0.739	± 0.033	65.639	± 5.273

Source: The Author.

Table 10 – Time values by the BCV algorithm cycles and foods in the urban scenario.

$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$	**Algorithm Time (s)**	**CI**	**Reduction in Execution Time (%)**	**CI**
20	20	0.029	± 0.008	61.442	± 6.252
20	50	0.054	± 0.009	63.367	± 4.820
20	100	0.108	± 0.011	64.945	± 4.359
50	20	0.052	± 0.008	64.423	± 4.861
50	50	0.136	± 0.013	65.228	± 4.894
50	100	0.292	± 0.022	65.156	± 4.815
100	20	0.119	± 0.012	64.367	± 5.154
100	50	0.287	± 0.023	66.769	± 4.601
100	100	0.589	± 0.043	62.300	± 5.621

Source: The Author.

Statistical Tests

BCV executions under different combinations of η_{cycles} and η_{foods} generated values for two performance metrics: "algorithm time" and "reduction in execution time". The different combinations have been tested under the same conditions. Nevertheless, it was necessary to carry out additional statistical tests due to overlapping confidence intervals (as seen in Tables 9 and 10). The Shapiro-Wilk test revealed that the data obtained is not normally distributed. Then, we performed Friedman and Wilcoxon tests to check if there is a significant difference between the combinations of cycles and food sources using the two metrics (SHESKIN, 2000).

For the first metric, each Friedman test compared the "algorithm time" for the different combinations of η_{cycles} and η_{foods} in highway and urban scenarios. The null hypothesis ($H0$) was that all combinations of η_{cycles} and η_{foods} have the same performance for the "algorithm time" metric. However, in all Friedman tests, we obtained a $p - value < 0.05$. Thus, $H0$ was rejected.

Then, the Wilcoxon test was applied to determine which samples' pairs have signifi-

cant differences in highway and urban scenarios. The test was done by comparing two-by-two samples. The null hypothesis ($H0$) of this test is that the corresponding sets of combinations η_{cycles} and η_{foods} are the same. This hypothesis ($H0$) was rejected in most cases (66/72) with a $p-value < 0.05$, meaning that most combinations are statistically different. The cases that $H0$ was not rejected with $p-value \geq 0.05$ are presented in Table 11. These cases have statistically the same performance.

Table 11 – Wilcoxon tests $p-values \geq 0.05$ for "algorithm time" metric.

Combination 1			**Combination 2**		**p-values**	
$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$		$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$	**Highway**	**Urban**
20	50	vs.	50	20	1.000	1.000
20	100	vs.	100	20	1.000	0.331
50	100	vs.	100	50	0.105	1.000

Source: The Author.

For the second metric, we performed Friedman tests that compared the metric "reduction in execution time" for the different combinations of η_{cycles} and η_{foods} in highway and urban scenarios. The null hypothesis of the tests ($H0$) states that the performance of reduction in execution time is the same for all combinations of η_{cycles} and η_{foods}. In the highway scenario, the p-value was 0.543. Thus $H0$ was not rejected, and the different combinations of η_{cycles} and η_{foods}, statistically, had the same performance concerning the "reduction in execution time" metric. In the urban scenario, $H0$ was rejected ($p-value < 0.05$).

Next, we applied the Wilcoxon test, making pairwise comparisons for the urban scenarios. The $H0$ of the test states that the different combinations η_{cycles} and η_{foods} have the same performance in the "reduction in execution time" metric. This hypothesis was not rejected in most cases (32/36). The cases that $H0$ was rejected are shown in Table 12. These cases show combinations of η_{cycles} and η_{foods} that have statically different performances concerning the "reduction in execution time" metric.

Table 12 – Wilcoxon tests $p-values < 0.05$ for "reduction in execution time" metric.

Combination 1			**Combination 2**		**p-values**
$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$		$\eta_{\textbf{cycles}}$	$\eta_{\textbf{foods}}$	**Urban**
20	20	vs.	50	50	0.016
20	20	vs.	100	50	0.007
20	50	vs.	100	50	0.015
50	20	vs.	100	50	0.017

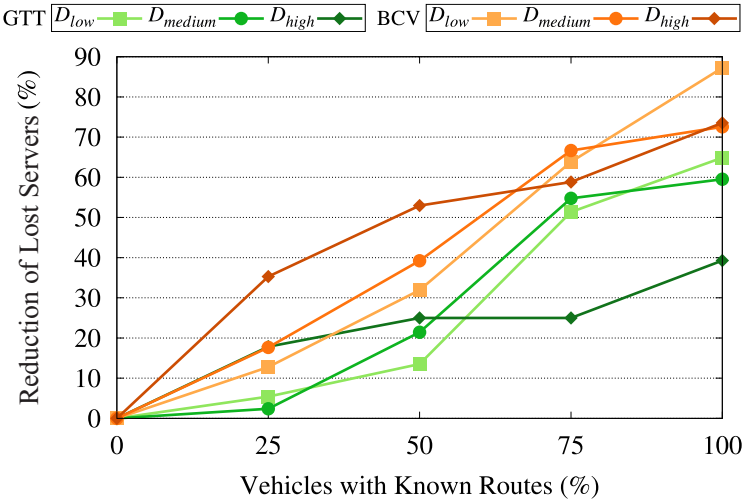
Source: The Author.

6.2.3 Known Routes of Vehicles

As mentioned earlier, some vehicles can share their routes with other nodes in the network. To analyze the impact of this information shared in computation offloading, we used the urban scenario for having more different route options. Besides, we used the workload with 12 tasks and the algorithms which use information about known routes (i.e., GTT and BCV). Such a number of tasks was chosen because distributing more tasks increases the chance of failure.

We tested 250 times each combination of density/algorithm/percentage of known routes of vehicles to account for the number of lost servers. A server is considered lost when it does not return all the results it owed to the *client*, causing failures in the computation offloading process. The experiment's benchmark is the number of lost servers when the percentage of vehicles with known routes is zero. Thus, we can see in Figure 17 that the more vehicles with known routes (x-axis), the greater the reduction of lost servers (and consequently the number of computation offloading failures).

Figure 17 – Reduction of lost servers for different percentages of known routes of vehicles.



Source: The Author.

For example, with 50 % of vehicles with known routes, GTT and BCV reduced by 25.0 % and 52.9 % (with D_{high}), respectively, the number of lost servers. The case of having 100 % of vehicles with known routes helped the GTT and BCV algorithms to reduce the number of lost servers by 64.8 % and 87.2 % (with D_{low}), respectively.

6.2.4 Discussion

Section 6.2.1 presented an analysis of results regarding the impact of the WAVE communication range on the request/reply process of the proposed framework. More specifically, the analysis was made by the average number of replies received by the *client* after it requested information from all vehicles within its range. We observe that, in general, the average number of replies received by the *client* is greater when the WAVE range is also greater. This is because the *client*'s request message can reach more vehicles, including the most distant ones. In turn, more consulted vehicles can send replies to the *client*. The only exception, in which there was no significant difference in the average number of replies, was when increasing the WAVE range from 500 to 750 m in the highway scenario with D_{medium}. In this case, due to the spatial distribution of vehicles in the scenario, the areas covered by the 500 and 750 m range of *clients* had practically the same number of vehicles.

In Section 6.2.1, another important analysis is that the increase in the density of vehicles from D_{medium} to D_{high}, mainly in the urban scenario, did not result in a significant difference in the average number of replies received by the *client*. This happened because, with high vehicular density (D_{high}), more vehicles of similar distances tried to send replies at the same time to the *client*. This simultaneous sending generated problems of contention and collision in the wireless medium. Thus, even though more vehicles sent replies, some of these replies failed to reach the *client*.

In Section 6.2.2, it is possible to analyze how the metrics "algorithm time" and "reduction in execution time" are impacted according to the two most important parameters of the BCV algorithm. These parameters are the number of cycles (η_{cycles}) and the number of foods (η_{foods}) (KARABOGA; AKAY, 2009; KHOSRAVANIAN *et al.*, 2018). They can be configured and combined in different ways. In general, we could observe that the higher the values of η_{cycles} and η_{foods}, the greater the averages of the "algorithm time" metric, i.e., the time for the BCV to choose the final solution. However, the different combinations used of η_{cycles} and η_{foods} resulted in the same performance concerning the "reduction in execution time" in the highway scenario and most cases in the urban scenario. This means that, statistically, the fitness of the solutions chosen by the BCV was the same, regardless of the combinations used of η_{cycles} and η_{foods}. In this way, for most cases, we can choose any combinations of the two parameters because the reduction in execution time of the workloads will not be altered.

We can analyze the impact of known routes of vehicles on computation offloading

processes in Section 6.2.3. Even with different vehicular densities, the known routes of vehicles reduced the number of lost servers, reducing offloading failures. This reduction occurs because this contextual information helps the GTT and BCV to better predict the network nodes' position at a given time. Consequently, these algorithms are able to calculate more precisely on which server each task can be executed so that failures/recoveries do not happen. In the assessed scenario, the BCV task allocation calculations generally result in greater reductions in lost servers than the GTT calculations. The explanation for this is that, unlike the GTT, the BCV also has evaluations of different task allocation solutions and opts for the most appropriate and reliable.

Values of the Parameters Used in the Performance Evaluation of Algorithms

For the performance evaluation of algorithms (Section 6.3), it was necessary to choose the official values of the parameters: WAVE communication range, number of cycles and food sources of the BCV algorithm, and percentage of vehicles with known routes. Although having a greater WAVE range generates a greater number of replies, offering more server options for clients, we defined the value of the range according to the literature. In this way, we use a WAVE range of 250 m. This value is often used in experiments and allows the IEEE 802.11p protocol to have a good performance (ARENA *et al.*, 2020; EICHLER, 2007). In addition, it is a range commonly used for file sharing (SOMMER; DRESSLER, 2014). About the number of cycles and food sources of the BCV algorithm, we use the following configuration. If the *client* had 1 GHz CPU, we made $\eta_{cycles} = 50$ and $\eta_{foods} = 50$. If the *client* had 0.5 GHz CPU, we made $\eta_{cycles} = 20$ and $\eta_{foods} = 50$. Regarding information on known routes, we have defined that 50 % of all vehicles in any scenario have complete routes to their respective destinations and can share information about those routes.

6.3 Performance Evaluation of Algorithms

This section contains the most important results and evaluations of this thesis. Section 6.3.1 presents data and discussions of an algorithm reliability metric called "Tasks by Occurrence Type". Through this metric, we detect how many failures and successes have occurred in the computation offloading processes. Section 6.3.2 describes results and evaluations of the "Reduction in Execution Time" metric. This metric is the main metric for evaluating the

performance of the algorithms. The data in Sections 6.3.1 and 6.3.2 were obtained through 200 simulations for each workload/density/scenario/algorithm configuration. In these sections, the experiments were presented in two parts. The first part deals with the execution of workloads T_1 to T_5 and the second part deals with the execution of workloads T_6 to T_{10}. Finally, the discussions regarding the analysis of results are presented in Section 6.3.3.

6.3.1 Tasks by Occurrence Type

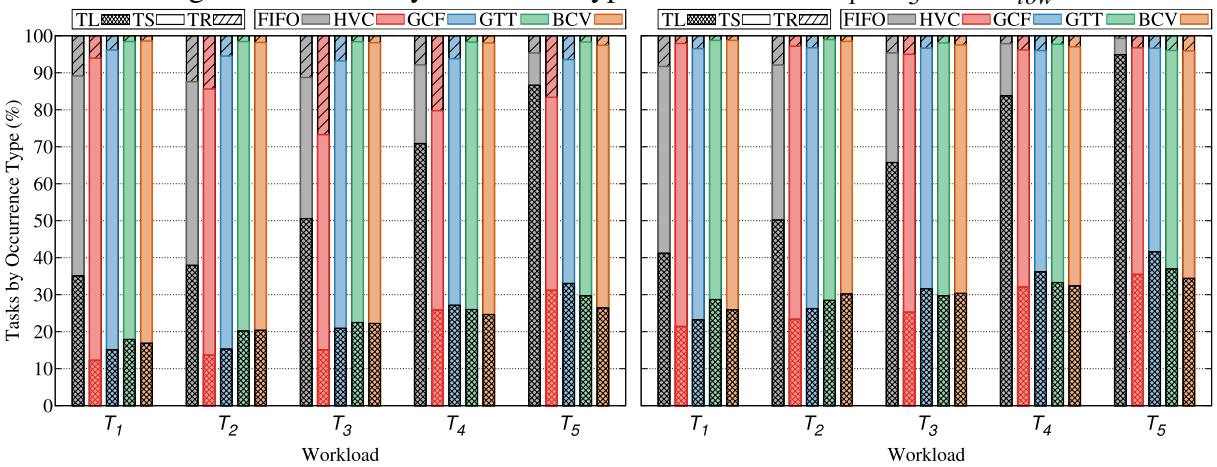
This section presents the performance of the proposed algorithms and others from the literature concerning what happens with each task of the different workloads. In this experiment, as shown in Figures 18 to 23, a task can have three occurrence types: TL, TS, or TR. TL (parts of the bars with checkered lines) represents tasks executed locally from the beginning by the *client*. TS (part of the bars without lines) represents tasks that were successfully offloaded, executed remotely, and the results were returned to the *client*. TR (parts of the bars with simple diagonal lines) represents tasks that were offloaded, and there was some failure in the process, causing them to need recovery on the *client*, as seen in Section 5.1.2. The percentages were calculated based on the sum of all occurrences for the 200 simulations for each workload/density/scenario/algorithm configuration. Next, we can analyze the results of these experiments.

Workloads T_1 to T_5 - Equal Tasks

In Figures 18 to 20, we have the percentage of tasks by occurrence type of the FIFO, HVC, GCF, GTT, and BCV algorithms with workloads T_1 to T_5. The data relating to the low vehicular density scenarios (D_{low}) are presented in Figures 18a and 18b. In the highway scenario (Figure 18a), BCV had the highest average percentage of TS (76.0 % - average between the five workloads analyzed), even offloading up to 81.7 % of tasks successfully with T_1. GTT had the lowest average percentage of TR (1.6 %). In the urban scenario (Figure 18b), the highest average percentage of TS was of the HVC (69.1 %). The lowest average percentage from TR was of the GTT (2.1 %), presenting only 1.0 % failed tasks with T_2.

Figures 19a and 19b show the data related to the medium density scenarios (D_{medium}). In the highway scenario (Figure 19a), the highest average percentage of TS was of the GTT (88.5 %), reaching an offloading success percentage of 90.0 % with T_2. The lowest average percentage of TR was of the BCV (0.6 %), presenting even a scenario without failures with T_1. In the urban scenario (Figure 19b), the GTT had the highest average percentage of TS (84.3 %) and the lowest

Figure 18 – Tasks by occurrence type for workloads $T_1 - T_5$ with D_{low}.

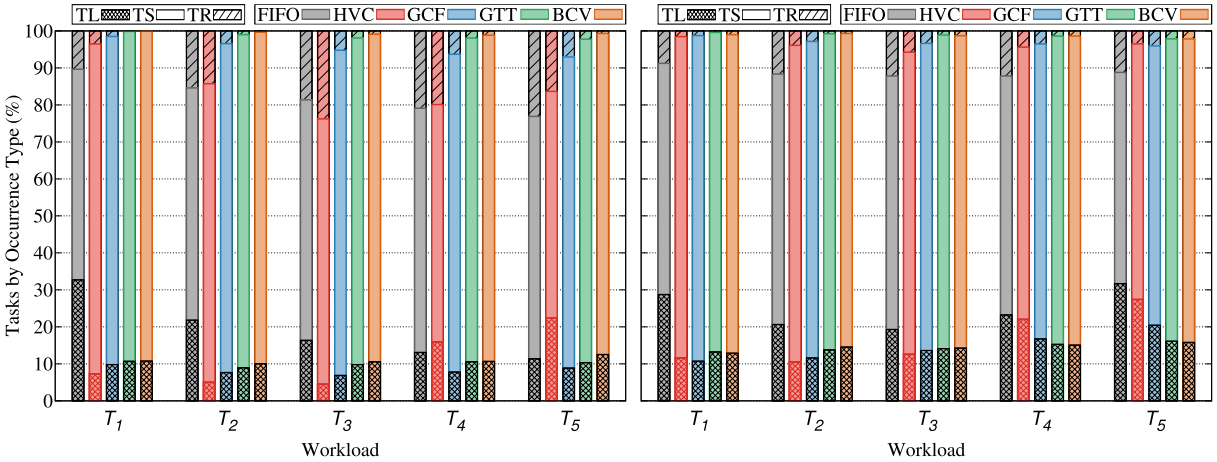


(a) Highway scenario. (b) Urban scenario.

Source: The Author.

percentage of TR (1.1 %).

Figure 19 – Tasks by occurrence type for workloads $T_1 - T_5$ with D_{medium}.

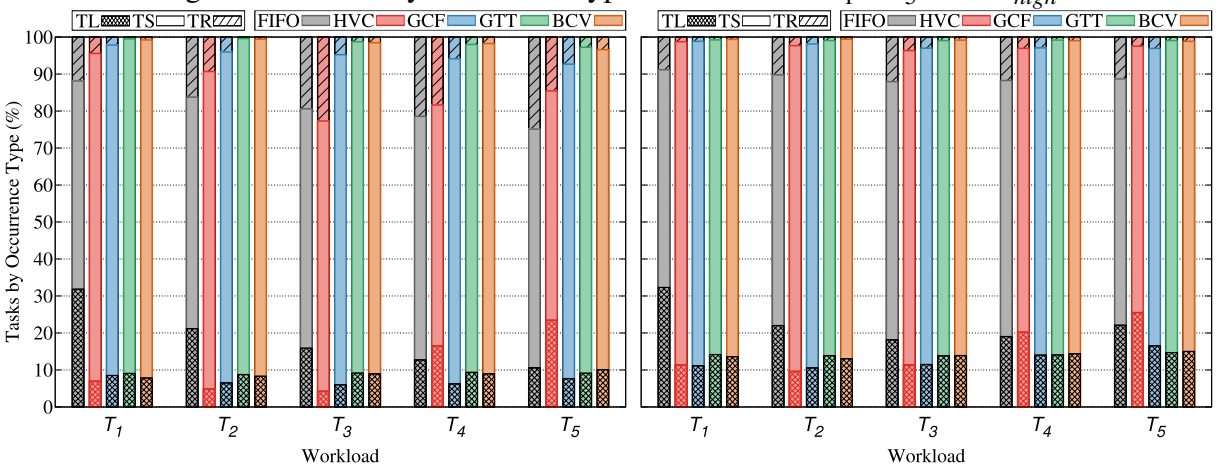


(a) Highway scenario. (b) Urban scenario.

Source: The Author.

The data for the high vehicular density scenarios (D_{high}) are presented in Figures 20a and 20b. In the highway scenario (Figure 20a), the GTT and BCV tied with the highest average percentage of TS (89.5 %). However, the BCV had the highest success rate (91.4 % with T_1). The lowest average percentage of TR was of the GTT (1.3 %), registering 0.4 % of failures with T_2. In the urban scenario (Figure 20b), the BCV had the highest average percentage of TS (85.1 %). The GCF had the highest success rate (87.7 % with T_1). In the lowest average percentage of tasks with failures, there was a tie between the GTT and BCV. Both had 0.8 %, with the BCV having the lowest rate of TR (0.5 % with T_2).

Figure 20 – Tasks by occurrence type for workloads $T_1 - T_5$ with D_{high}.



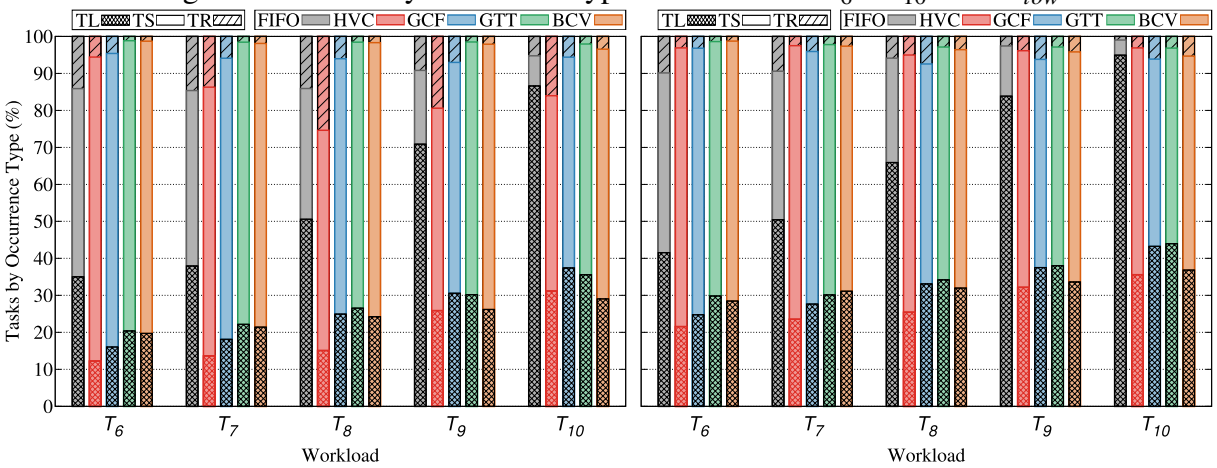
(a) Highway scenario. (b) Urban scenario.

Source: The Author.

Workloads T_6 to T_{10} - Different Tasks

Figures 21 to 23 show the percentages of tasks by occurrence type of the FIFO, HVC, GCF, GTT, and BCV algorithms with the workloads T_6 to T_{10}. The data relating to low vehicular density scenarios (D_{low}) are shown in Figures 21a and 21b. In the highway scenario (Figure 21a), the algorithm that achieved the best average percentage of TS was the BCV (73.8 %). The GTT had the best average percentage of TR (1.5 %), even having a case that only 1.1 % of the tasks failed (with T_6). In the urban scenario (Figure 21b), the HVC achieved the highest average percentage of TS (68.8 %), and the GTT had the lowest percentage of TR (2.4 %).

Figure 21 – Tasks by occurrence type for workloads $T_6 - T_{10}$ with D_{low}.

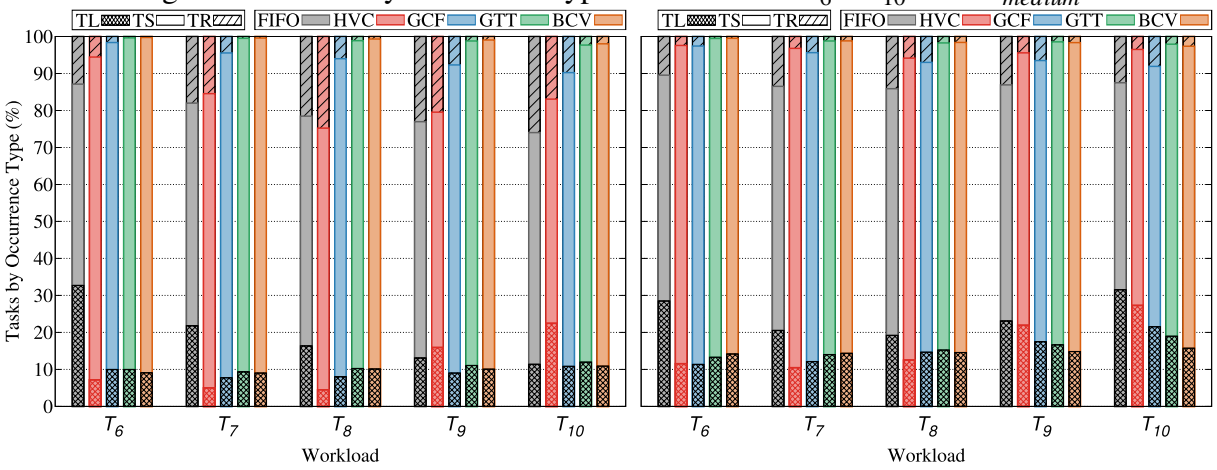


(a) Highway scenario. (b) Urban scenario.

Source: The Author.

The data for the scenarios with D_{medium} are shown in Figures 22a and 22b. In the highway scenario (Figure 22a), the BCV achieved the highest average TS rate (89.3 %), even managing to offload 90.6 % of tasks successfully (with T_6). In terms of recovered tasks, the BCV also achieved the best performance. It achieved an average TR percentage of 0.8 %, having the lowest failure rate in a specific case (0.2 % with T_6). In the urban setting (Figure 22b), the BCV achieved the highest average percentage of TS (83.7 %). The lowest average percentage of TR was of the GTT (1.4 %), with the BCV and GTT tying at the lowest failure rate for a specific case (0.5 % with T_6).

Figure 22 – Tasks by occurrence type for workloads $T_6 - T_{10}$ with D_{medium}.



(a) Highway scenario. (b) Urban scenario.

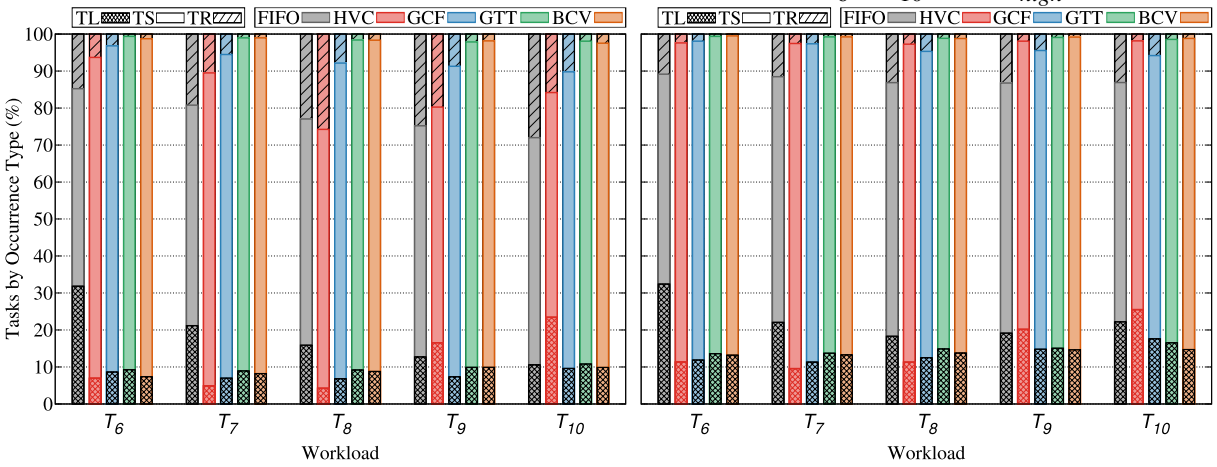
Source: The Author.

Finally, the data for D_{high} are detailed in Figures 23a and 23b. In the highway scenario (Figure 23a), the algorithm with the best performance in the average success rate was the BCV (89.5 %), reaching 91.3 % of TS. The algorithm with the lowest average percentage of TR was the GTT, achieving that only 0.6 % of the tasks failed with T_6. In the urban scenario (Figure 23b), the BCV achieved the highest average percentage of TS (85.1 %) and the lowest average percentage of TR (0.8 %), with failures in only 0.5 % of tasks (T_6 case).

Results Summary

Based on Figures 18 to 23, Tables 13 and 14 present a summary of the "tasks by occurrence type" metric results. The "TS > FIFO & TS > HVC" and "TR < FIFO & TR < HVC" columns contain, respectively, the number of times that the percentage of TS and TR of a proposed algorithm was better than that of the algorithms FIFO and HVC. This comparison is

Figure 23 – Tasks by occurrence type for workloads $T_6 - T_{10}$ with D_{high}.



(a) Highway scenario. (b) Urban scenario.

Source: The Author.

made with a group of 30 experiments (scenario/density/workload). The "Record TS" and "Record TR" columns contain the highest TS and the lowest TR values of the algorithms proposed for the group of 30 experiments. The "Best TS Average" and "Best TR Average" columns use a group of experiments of six combinations since the averages of the five workloads of each scenario/density are used. The "Best TS Average" and "Best TR Average" contain, respectively, the number of times that a proposed algorithm had the best average of TS and TR among all the algorithms of each scenario/density.

According to Table 13, with the workloads T_1 to T_5, the BCV more often outperformed the FIFO and HVC algorithms in percentages of TS (22/30) and TR (28/30, tied with the GTT). It also registered the record of TS (91.4 %) and TR (0.0 %) and presented the best TS averages (3/6, tied with the GTT) more often. In turn, the GTT registered best TR averages more frequently (5/6).

Table 13 – Results summary of Section 6.3.1 with workloads T_1 to T_5.

Algorithm	TS > FIFO & TS > HVC	Record TS	Best TS Average	TR < FIFO & TR < HVC	Record TR	Best TR Average
BCV	22/30	91.4 %	3/6	28/30	0.0 %	2/6
GTT	20/30	90.8 %	3/6	28/30	0.1 %	5/6
GCF	21/30	89.4 %	0/6	23/30	1.1 %	0/6

Source: The Author.

With the T_6 to T_{10} workloads, as shown in Table 14, the BCV had the TS record (91.3 %), the TR record (0.2 %), and the highest number of times with the best TS average (5/6). The GTT was the one that more frequently outperformed the FIFO and HVC with the best percentage

of TS (20/30) and TR (28/30). In addition, GTT had the highest number of times with the best TR average (4/6).

Table 14 – Results summary of Section 6.3.1 with workloads T_6 to T_{10}.

Algorithm	TS > FIFO & TS > HVC	Record TS	Best TS Average	TR < FIFO & TR < HVC	Record TR	Best TR Average
BCV	19/30	91.3 %	5/6	27/30	0.2 %	2/6
GTT	20/30	90.1 %	0/6	28/30	0.3 %	4/6
GCF	18/30	88.3 %	0/6	15/30	1.6 %	0/6

Source: The Author.

6.3.2 Reduction in Execution Time

This section analyzes the algorithms' impact in reducing the execution time of different workloads compared to executing all tasks locally on the client (benchmark). The averages were calculated based on the values obtained in the simulations for each workload/density/scenario/algorithm configuration. The Figures' error bars show the 95% confidence interval of the corresponding data. The absolute values of the execution times can be obtained according to the calculations of local computation time (benchmark) presented in Section 4.1.3 for the different workloads. The following paragraphs show the results of this experiment.

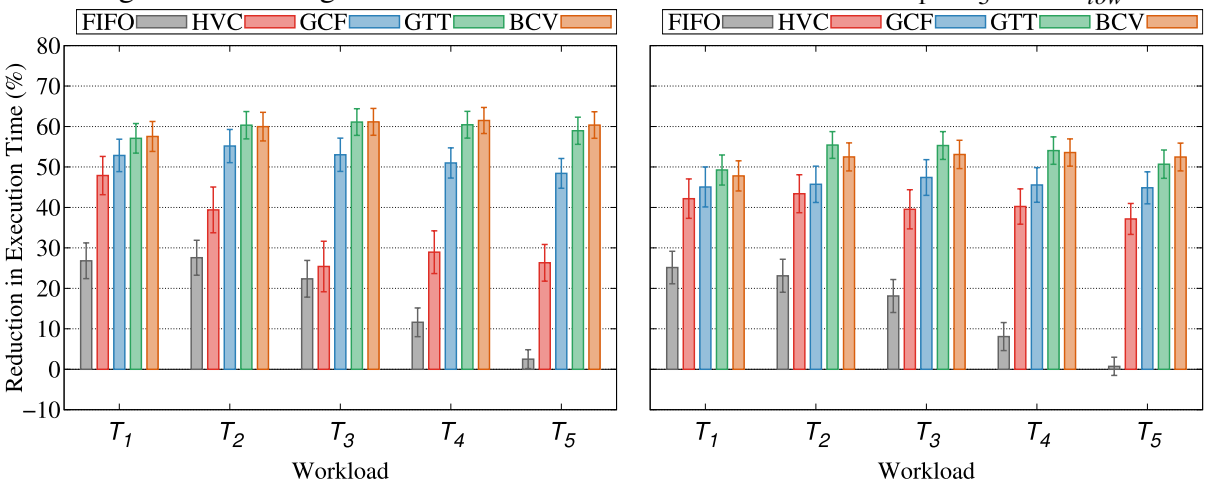
Workloads T_1 to T_5 - Equal Tasks

Figures 24 to 26 show the average percentages of reductions in the execution time of the workloads T_1 to T_5 for the different algorithms. Experiments of scenarios with D_{low} are shown in Figures 24a and 24b. In the highway scenario (Figure 24a), the greatest average reductions in execution time were of the BCV and GTT. The BCV had the highest average reduction (61.4 % with T_4). In the urban scenario (Figure 24b), the GTT and BCV also had the best performances. The highest average reduction was of the GTT (55.4 % with T_2).

The medium density scenarios are shown in Figures 25a and 25b. In the highway scenario (Figure 25a), the BCV and GTT achieved the highest averages of reductions. The GTT achieved the largest average reduction (72.8 % with T_5). In the urban scenario (Figure 25b), the best averages of reductions were also of the BCV and GTT. GTT guaranteed the most significant average reduction (67.5 % with T_5).

Figures 26a and 26b show the data related to the scenarios with D_{high}. In the highway scenario (Figure 26a), the largest averages of reductions were of the BCV and GTT. The GTT

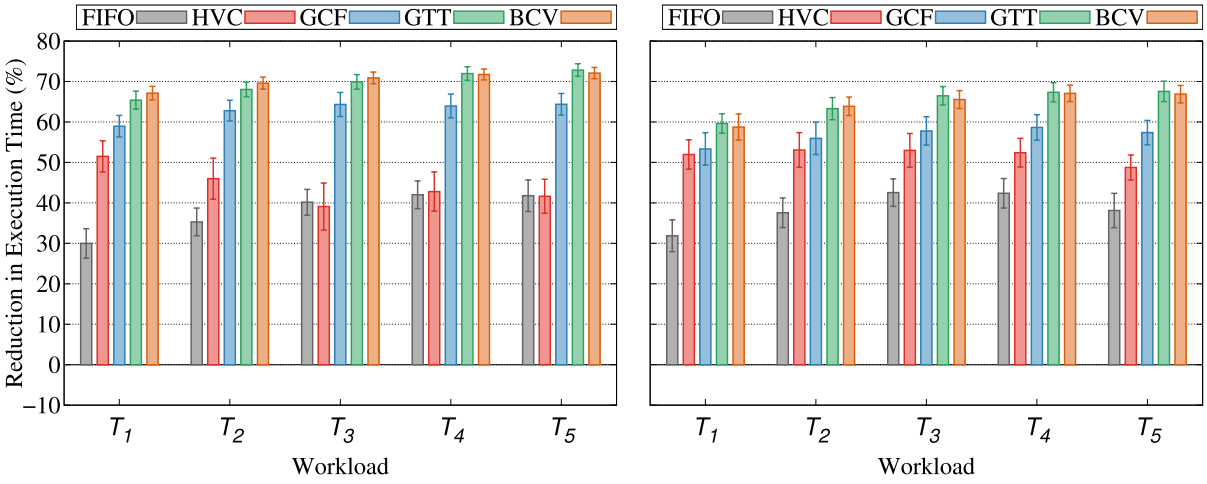
Figure 24 – Average reduction in execution time for workloads $T_1 - T_5$ with D_{low}.



(a) Highway scenario. (b) Urban scenario.

Source: The Author.

Figure 25 – Average reduction in execution time for workloads $T_1 - T_5$ with D_{medium}.



(a) Highway scenario. (b) Urban scenario.

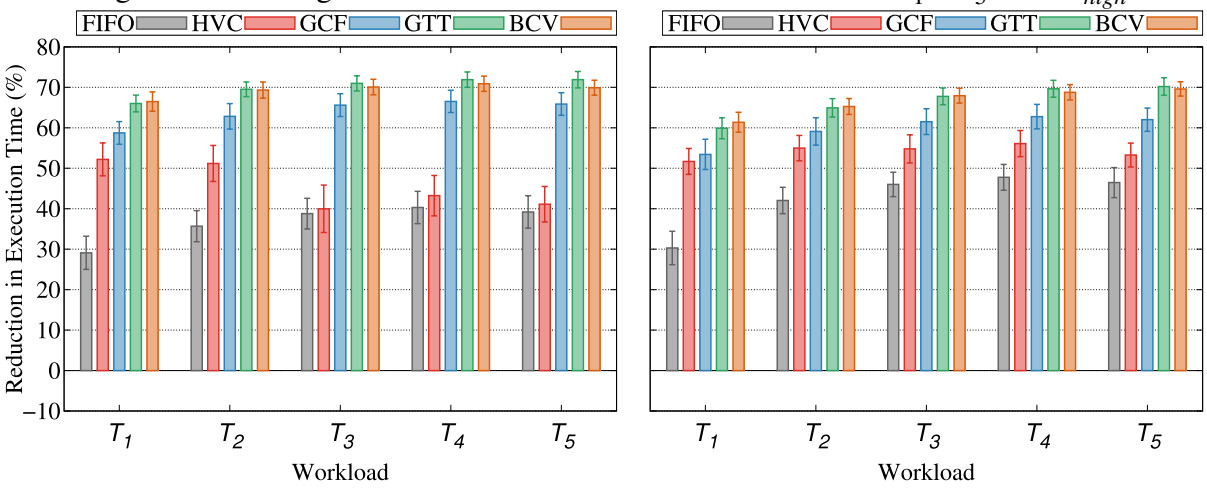
Source: The Author.

presented the most considerable average reduction (71.9 % with T_4 and T_5). In the urban scenario (Figure 26b), the BCV and GTT also obtained the best averages of reductions. GTT presented the highest average reduction for a specific workload (70.2 % with T_5).

Statistical Tests

We perform additional statistical tests to evaluate the algorithms' performance better when overlapping confidence intervals exist between them. When performing the Shapiro-Wilk tests, we realized that the data from these experiments are not normally distributed. Such data were obtained through repeated experiments under the same conditions. Thus, we used the

Figure 26 – Average reduction in execution time for workloads $T_1 - T_5$ with D_{high}.



(a) Highway scenario.

(b) Urban scenario.

Source: The Author.

Friedman test to assess statistical differences in the 30 groups of experiments (scenario/density/-workload) (SHESKIN, 2000). The null hypothesis ($H0$) is that the performance of reductions in the execution time is the same for all algorithms. We obtained $p - values < 0.05$ in all 30 groups, rejecting the null hypothesis.

Then we made pairwise comparisons between the algorithms using the Wilcoxon test[3]. In this test, the null hypothesis ($H0$) states that the compared algorithms have the same performance. If $p - value < 0.05$, we reject $H0$ and consider that the compared algorithms have statistically different performances. The p-values of these tests for workloads T_1 to T_5 are shown in Tables 15 to 17.

Table 15 presents the p-values of the Wilcoxon tests for pairwise comparisons between algorithms with low vehicular density in highway and urban scenarios. We can observe that the BCV and GTT have the same performance ($p - value > 0.05$) in 4/5 cases with the highway scenario and in 2/5 cases with the urban scenario.

The p-values of the Wilcoxon tests for the paired comparisons of algorithms for medium density scenarios are shown in Table 16. Counting the highway and urban scenarios, the GTT and BCV have the same performance ($p - value > 0.05$) in 6/10 cases.

Finally, Table 17 shows the p-values of the Wilcoxon tests for high vehicular density scenarios. In the highway scenario, the BCV and GTT have the same performance of reduction in execution time in just one case (with T_2). In the urban scenario, the BCV and GTT have the

[3] We have not made pairwise comparisons with the FIFO algorithm because it does not have overlapping CIs with the proposed algorithms GCF, GTT and BCV.

Table 15 – Wilcoxon tests p-values for workloads T_1 to T_5 with D_{low}.

	Highway					Urban				
Algorithms	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**
BCV vs. GTT	1.000	1.000	1.000	0.101	0.039	0.000	0.000	0.006	1.000	0.705
BCV vs. GCF	0.058	0.140	0.000	0.000	0.000	1.000	0.000	0.047	0.000	0.000
BCV vs. HVC	0.000	0.000	0.000	0.000	0.000	0.014	0.000	0.000	0.000	0.000
GTT vs. GCF	0.873	0.397	0.000	0.000	0.000	0.034	0.000	0.000	0.000	0.000
GTT vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GCF vs. HVC	0.129	0.000	0.000	0.000	0.000	0.859	0.259	0.000	0.000	0.000

Source: The Author.

Table 16 – Wilcoxon tests p-values for workloads T_1 to T_5 with D_{medium}.

	Highway					Urban				
Algorithms	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**
BCV vs. GTT	0.808	0.348	1.000	0.035	0.000	1.000	0.694	0.036	0.102	0.000
BCV vs. GCF	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. GCF	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GCF vs. HVC	0.000	0.000	0.000	0.000	0.000	1.000	0.053	0.011	0.000	0.000

Source: The Author.

same performance in 3/5 cases (T_1 to T_3).

Table 17 – Wilcoxon tests p-values for workloads T_1 to T_5 with D_{high}.

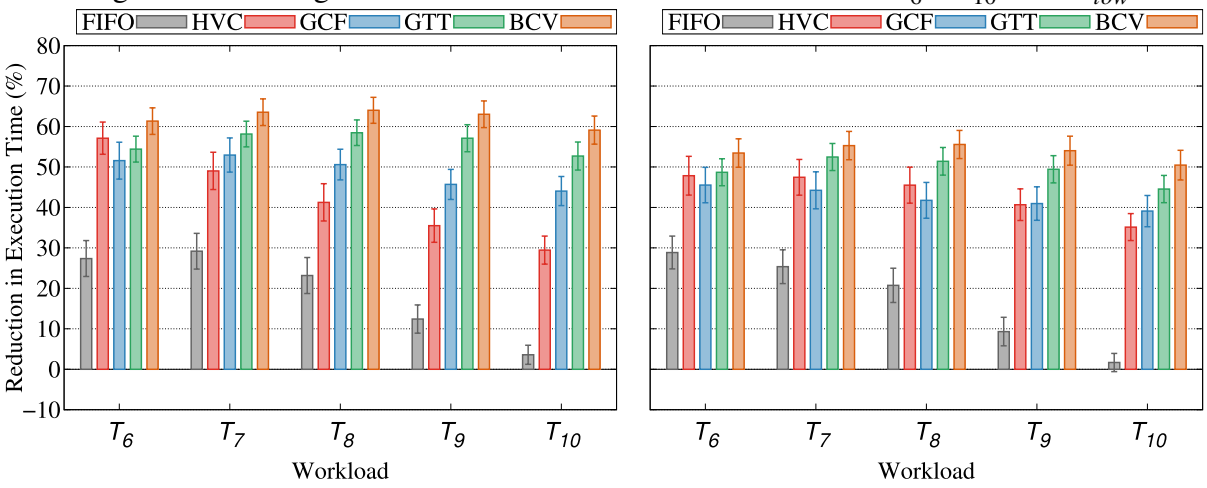
	Highway					Urban				
Algorithms	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**
BCV vs. GTT	0.007	1.000	0.025	0.007	0.001	0.090	0.890	0.231	0.000	0.002
BCV vs. GCF	0.000	0.000	0.028	0.036	0.039	0.000	0.000	0.000	0.000	0.000
BCV vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. GCF	0.000	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GCF vs. HVC	0.003	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000

Source: The Author.

Workloads T_6 to T_{10} - Different Tasks

According to Figures 27 to 29, we present the data of the reductions in the execution time of the workloads T_6 to T_{10} for the FIFO, HVC, GCF, GTT, and BCV algorithms. Figures 27a and 27b show the data related to scenarios with D_{low}. In the highway scenario (Figure 27a), the BCV algorithm obtained the highest averages of reductions. It also recorded the highest average reduction in execution time (64 % with T_8). In the urban scenario (Figure 27b), the BCV also presented the best averages of reductions in execution time and marked the most significant reduction in execution time (55.5 % with T_8).

Figure 27 – Average reduction in execution time for workloads $T_6 - T_{10}$ with D_{low}.

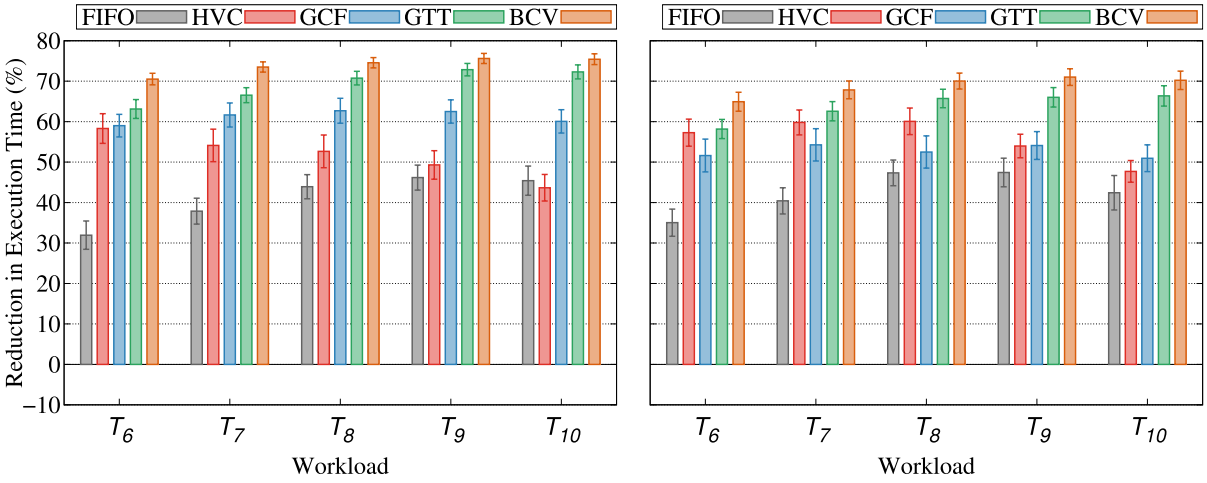


(a) Highway scenario. (b) Urban scenario.

Source: The Author.

Figures 28a and 28b present the data related to the scenarios with D_{medium}. In the highway scenario (Figure 28a), the best average reductions in execution time were of the BCV. It also recorded the largest average reduction (75.6 % with T_9). In the urban scenario (Figure 28b), the BCV presented the best performance and obtained the best average reduction for a workload (71.0 % with T_9).

Figure 28 – Average reduction in execution time for workloads $T_6 - T_{10}$ with D_{medium}.



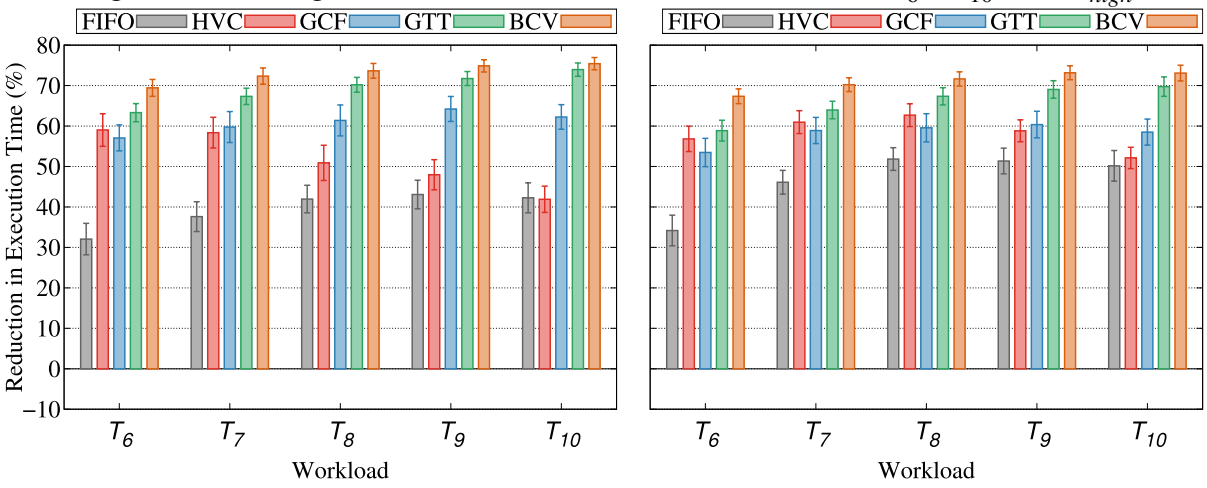
(a) Highway scenario. (b) Urban scenario.

Source: The Author.

The data relating to the scenario with D_{high} are shown in Figures 29a and 29b. In the highway scenario (Figure 29a), the BCV algorithm showed the best averages of execution time reductions. In addition, it recorded the greatest average reduction for a workload (75.4 % for T_{10}). In the urban scenario (Figure 29b), the BCV also achieved the best performance and the

most significant average reduction (73.1 % for T_9).

Figure 29 – Average reduction in execution time for workloads $T_6 - T_{10}$ with D_{high}.



(a) Highway scenario.

(b) Urban scenario.

Source: The Author.

Statistical Tests

The data observed in Figures 27 to 29 have overlapping CIs and were generated through experiments under the same conditions. According to the Shapiro-Wilk tests we have made, they are not normally distributed. Then, we used the Friedman test to detect statistical differences between the performance of the algorithms (SHESKIN, 2000). The null hypothesis ($H0$) tested states that the performance of all algorithms is the same for the metric "reduction in execution time". In all tests, the p-value was less than 0.05, rejecting $H0$.

Then, we perform Wilcoxon tests to determine which pairs of algorithms have different performances from each other[4]. The null hypothesis ($H0$) for each test is that the pair of algorithms have the same performance in the analyzed metric. If $p - value < 0.05$, we reject $H0$ and consider the performance of the algorithms to be statically different. The p-values of these tests for the workloads T_6 to T_{10} are presented in Tables 18 to 20.

In Table 18, we can see the p-values of the Wilcoxon tests to compare pairs of algorithms with the low vehicular density scenarios. It is also possible to observe that the BCV and GTT do not have the same performance in either case.

Table 19 presents the p-values of the Wilcoxon tests for pairwise comparisons of the

[4] The FIFO was not compared because there is little or no overlap of confidence interval with the proposed algorithms.

Table 18 – Wilcoxon tests p-values for workloads T_6 to T_{10} with D_{low}.

	Highway					Urban				
Algorithms	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**
BCV vs. GTT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. GCF	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. HVC	0.195	0.000	0.000	0.000	0.000	0.021	0.000	0.000	0.000	0.000
GTT vs. GCF	1.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
GTT vs. HVC	0.007	0.008	0.000	0.000	0.000	1.000	0.202	0.023	0.000	0.000
GCF vs. HVC	0.000	0.887	0.004	0.000	0.000	0.015	0.052	0.119	1.000	0.002

Source: The Author.

HVC, GCF, GTT, and HVC algorithms with medium vehicular density. Again, the BCV and GTT do not have the same performance in either case ($p - value = 0.000$ in all cases).

Table 19 – Wilcoxon tests p-values for workloads T_6 to T_{10} with D_{medium}.

	Highway					Urban				
Algorithms	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**
BCV vs. GTT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. GCF	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. GCF	0.036	0.111	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000
GTT vs. HVC	0.873	0.000	0.000	0.000	0.000	0.608	1.000	0.037	0.000	0.000
GCF vs. HVC	1.000	0.027	0.001	0.000	0.000	0.000	0.000	0.000	1.000	0.189

Source: The Author.

Finally, Table 20 shows the p-values of the comparisons between the algorithms using the Wilcoxon test with scenarios of high vehicular density. Again, the two best algorithms (BCV and GTT) showed a statistically significant difference in all cases.

Table 20 – Wilcoxon tests p-values for workloads T_6 to T_{10} with D_{high}.

	Highway					Urban				
Algorithms	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**	**T_6**	**T_7**	**T_8**	**T_9**	**T_{10}**
BCV vs. GTT	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000
BCV vs. GCF	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BCV vs. HVC	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GTT vs. GCF	0.000	0.001	0.003	0.000	0.000	0.013	0.003	0.000	0.000	0.000
GTT vs. HVC	1.000	0.000	0.000	0.000	0.000	1.000	0.094	0.002	0.000	0.000
GCF vs. HVC	0.007	1.000	0.000	0.000	0.000	0.050	1.000	0.486	0.153	0.000

Source: The Author.

Results Summary

According to Figures 24 to 29 and Tables 15 to 20, Tables 21 and 22 summarize the "reduction in execution time" metric results. The column "RD > FIFO & RD > HVC" contains

the number of times that the average Reduction in the execution time (RD) of a proposed algorithm was greater than that of the FIFO and HVC algorithms. The columns "Record to FIFO" and "Record to HVC" indicate, respectively, the most significant difference in percentage points between a proposed algorithm and the FIFO and HVC algorithms. The "Record RD" column shows the highest average reduction in execution time for a proposed algorithm. Finally, the "Best Performance" column contains the number of times that one algorithm achieved the best performance among all other algorithms in the "reduction in execution time" metric.

According to Table 21, with the equal task workloads (T_1 to T_5), the BCV and GTT outperformed the FIFO and HVC by 30/30 times. Compared to the FIFO, the highest average reduction in execution time was of the BCV (57.9 % percentage points of difference). Compared to HVC, the record difference in the reduction in execution time was up to 35.7 % percentage points, achieved by the BCV and GTT. The GTT also recorded the most considerable average reduction (72.8 %). It was also the algorithm that had the best performance most times (28/30), but with a performance very similar to that of the BCV in most cases.

Table 21 – Results summary of Section 6.3.2 with workloads T_1 to T_5.

Algorithms	**T_1 to T_5**				
	RD > FIFO & RD > HVC	**Record to FIFO**	**Record to HVC**	**Record RD**	**Best Performance**
BCV	30/30	57.9 %	35.7 %	72.0 %	18/30
GTT	30/30	56.5 %	35.7 %	72.8 %	28/30
GCF	24/30	46.0 %	27.6 %	66.5 %	1/30

Source: The Author.

With different task workloads (T_6 to T_{10}), as shown in Table 22, BCV proved to be the best algorithm. It was the one that most often outperformed the FIFO and the HVC (29/30), registering the records of percentage differences for them in the metric reduction in the execution time (55.6 % and 33.6 %, respectively). The BCV also recorded the highest average reduction (75.6 %) and was the algorithm that obtained the best performance more times (30/30).

Table 22 – Results summary of Section 6.3.2 with workloads T_6 to T_{10}.

Algorithms	**T_6 to T_{10}**				
	RD > FIFO & RD > HVC	**Record to FIFO**	**Record to HVC**	**Record RD**	**Best Performance**
BCV	29/30	55.6 %	33.6 %	75.6 %	30/30
GTT	21/30	49.1 %	32.1 %	73.9 %	0/30
GCF	12/30	40.5 %	20.4 %	64.2 %	0/30

Source: The Author.

6.3.3 Discussion

According to the following paragraphs, this section reports on discussions about the implications of the "Tasks by Occurrence Type" and "Reduction in Execution Time" metrics. After, we present the main discussions about the results.

Implications of the "Tasks by Occurrence Type" Metric

Section 6.3.1 presents the analysis of the results of the "Tasks by Occurrence Type" metric. Three types of occurrences are possible to happen with a task: TL, TS, or TR. They represent, respectively, local execution, successful remote execution, and failure in the remote execution of tasks (with recovery and local execution of the lost task). The ideal result is that a computation offloading algorithm has a high TS rate and a low TR rate. This result would indicate that the algorithm is choosing to execute most tasks on reliable remote servers. In this way, the computation would be more distributed and the chosen servers would be less likely to lose connectivity with the *client*.

Moreover, having a high TS percentage is a good indicator because it relieves the client's processing overload and tasks can be executed more quickly on remote servers with better computing resources. A low TR percentage is also an important indicator. Sending tasks to be performed remotely involves using bandwidth and processing on the remote server. It also involves time to transmit, process, and wait for their results. When a failure occurs, these times and computational resources are wasted. Also, as the result of task processing does not arrive, it is necessary to recover the lost task. So, time is still needed to detect the failure and re-execute the task locally. Thus, a low TR percentage indicates a low percentage of failures, saving time and computational resources.

Implications of the "Reduction in Execution Time" Metric

Section 6.3.2 reports the analysis of the results of the "Reduction in Execution Time" metric. This metric compares the execution times of the workloads resulting from the allocation of the algorithms with the benchmark, i.e., the time it takes for the *client* vehicle to execute all tasks locally. The main objective of the algorithms is to reduce the execution time of vehicular applications as much as possible. The computation offloading processes of the algorithms distribute the computational tasks to other vehicles and edge servers so that this time is reduced.

In this respect, the most critical step is the task assignment, which involves deciding on which server each task will be executed. A bad decision can send tasks to servers that are already overloaded, have low processing capacity, are distant from the client, or quickly lose connectivity. This decision will not only not be able to reduce the execution time of vehicular applications, but it can even increase that time. On the other hand, a good decision allows to send tasks to nearby servers, not overloaded, with greater processing capacity, and that will remain connected with the *client* for a long period. This decision results in fewer failures and faster execution of computational tasks.

Impacts of the Scenario and Density on Results

In general terms, our urban scenarios, which have more sparsely located vehicles, have lower TS and RD rates and higher TL rates than our highway scenarios, whose vehicles are spatially more concentrated. Low vehicular density scenarios also have lower TS and RD rates and higher TL than medium or high vehicular density scenarios. This behavior in urban and low density scenarios is because fewer servers are available within the *client*'s WAVE range. Thus, the *client* needs to execute more tasks locally, becoming overloaded and delaying their execution. However, there are more server options for the *client* in higher vehicular density scenarios, especially in highway scenarios. So, the algorithms tend to choose more remote executions for the computational tasks, decreasing the TL rates and increasing the TS and RD rates.

Proposed Algorithms vs. Other Algorithms

Regarding the performance of the algorithms, we can see two main results about the proposed algorithms, GCF, GTT, and BCV, if compared to the FIFO and HVC in most cases. First, they showed higher TS rates and lower TR rates. Second, they have shown greater reductions in execution times, our main performance metric. In this way, the proposed algorithms help vehicular applications to execute their tasks more quickly and reliably, i.e., with a low TR rate.

One of the factors that explains the better performance of the proposed algorithms is the simultaneous use of the WAVE and 5G networks. This use allows the GCF, GTT, and BCV to take advantage of both the vehicular clouds (with V2V) and the edge servers (with V2I). It also allows to increase the transmission capacity of tasks and reduce latencies. Besides, it combines

the independence of infrastructure and direct communications from WAVE networks and the large data rates and increased band spectrum availability of 5G networks.

In addition, the strategy used by the proposed algorithms results in their better performance compared to the totally local execution and the FIFO and HVC algorithms. In this way, GCF, GTT, and BCV make better server choices for executing tasks. The choice prioritizes the servers closest to the *client*, with acceptable link lifetimes, and the lowest processing queue times.

GTT and BCV vs. Other Algorithms

However, among the three proposed algorithms, the best performances were of the GTT and BCV. One reason is that these two algorithms are the only ones to use the contextual information of CPU capacity of servers (in GHz) and known routes of vehicles. Thus, with more contextual information, the task assignments of these two algorithms become more accurate. The GTT's strategy of verifying the best possible server after each task allocation and the BCV's intelligent strategy based on ABC metaheuristics also improve task assignments. Consequently, GTT and BCV can make better and more reliable offloading decisions, choosing the best servers for each task. With this, servers can execute tasks faster and the results can be returned to the *client* before they lose connectivity, avoiding failures.

Information about known routes of vehicles also makes a special contribution to reducing the offloading processes' failures and task recoveries of the GTT and BCV algorithms. With fewer recoveries, the reduction in the total execution time of the workload is greater. The other algorithms, with more recoveries, end up delaying more their total execution time.

GTT vs. BCV

When comparing the BCV and GTT, it is also possible to see that they have a very similar performance in TS and TR rates. In terms of reduction in execution time, we split the evaluation depending on the workloads. With the equal task workloads (T_1 to T_5), although the two algorithms have very similar performances, the GTT stands out. As a greedy algorithm, it periodically updates the best possible server (including the client itself) to execute each task, taking advantage of the computational resources available from all nodes in the network. Furthermore, since the tasks are the same, the GTT makes the best allocations, just as the BCV does. However, the GTT task assignment requires a time of approximately 0 ms. On the other

hand, the BCV task assignment/schedule can take up to hundreds of milliseconds. The time it takes the BCV to choose the final solution impairs its performance in reducing execution time compared to the GTT.

With the different task workloads (T_6 to T_{10}), the BCV outperforms the GTT in reducing execution times in all scenarios. In such cases, the GTT continues to make reliable server choices, causing few offloading failures. However, these choices are less effective because the GTT greedily seeks the best servers for the tasks without considering their requirements. In this way, the GTT can send multiple tasks with low requirements of CPU cycles to the best server available. In the end, tasks with high CPU cycle requirements may be leftover. In that case, the best available server may already be overloaded with other tasks. So, the GTT may end up sending these more complex tasks to servers with lower computational capacities and longer processing queue times.

For workloads T_6 to T_{10}, although the BCV takes some time to choose the final solution, this choice is more intelligent and more effective. It analyzes the fitness of different solutions and chooses the most appropriate one, taking into account the best available servers and the characteristics of each task in the workload. Thus, the BCV has a similar performance to the GTT with the same tasks and outperforms it when the tasks are different.

Therefore, the GTT and BCV make more accurate contextual evaluations, better server choices, better task assignment, and takes advantage of the simultaneous use of WAVE, 5G, and the vehicular edge computing system. Through extensive simulations and diverse vehicular environments, we show that the GTT and BCV achieve the greatest reductions in the total execution times of tasks and the lowest rates of offloading failures, solving the Problem P1.

6.4 Concluding Remarks

This chapter described the configuration of several aspects, parameters, and scenarios used in the experiments. Then, the analysis and evaluation of results about important parameters of network, mobility, and the BCV algorithm were presented. For example, we saw that the greater the WAVE range, the greater the number of reply messages received by the *client*. The two most important parameters of the BCV algorithm were also evaluated: number of cycles and number of food sources. We saw that the greater they are, the longer it takes the algorithm to find the final solution. However, changing these two parameters is not so important for reducing workload execution times. We were also able to see that the higher the percentage of vehicles

with known and propagated routes, the greater the reduction in lost servers, resulting in fewer failures in the computation offloading processes.

Then, the performances of the different algorithms were evaluated according to the two main metrics of this thesis: "tasks by occurrence type" and "reduction in execution time". The first metric evaluates the percentage of successes and failures in the offloading processes. The second metric evaluates how much each algorithm has reduced in terms of the workload execution times when compared to the totally local execution by the *client*. We saw that the proposed algorithms outperform the totally local execution and the FIFO and HVC algorithms in the two metrics. In discussing the results, we evaluated the impact of scenarios and vehicle densities on the results. Finally, we present the reasons why the proposed algorithms have the best performances in the scenarios. From this evaluation, we were able to conclude that the GTT and BCV algorithms have the best performance for equal task workloads, and BCV has the best performance for different task workloads. Therefore, the GTT and BCV solved the Problem P1, presenting the best ways to reduce the execution time of vehicular applications reliably.

7 CONCLUSION

This chapter presents the latest discussions and insights from this work. Section 7.1 provides responses to the research questions listed in Chapter 1. Next, Section 7.2 discusses the objectives, contributions, importance, and implications of this thesis. Finally, Section 7.3 highlights research directions for future work.

7.1 Responses to Research Questions

This section presents the responses to the three research questions listed in Chapter 1 of this thesis. In the following paragraphs, the questions are re-presented, and the responses are explained.

RQ1: How to provide support and management for all stages of computation offloading processes?

We propose that the context-oriented framework designed and developed during the research of this thesis supports and manages all stages of computation offloading processes both on clients and servers. In the case of the client vehicle, when receiving tasks from an application, the framework starts discovering computational resources available. Through a request/reply process, the client vehicle becomes aware of several contextual parameters from possible servers. Then all task and server information is gathered in the decision module. After the decision, made by a task assignment algorithm, the framework supports and manages to send tasks to remote servers and receive the processing results. After receiving the results, the framework sends them to the application to continue its execution.

In the case of a server, the framework replies to client requests. Then it receives computation tasks to be processed if there is enough energy. After processing, the framework manages the step of sending processing results back to clients.

RQ2: How to assign computation tasks to different servers to reduce vehicular applications' execution time in VEC systems?

We propose that this assignment be done through one of the three decision and task assignment algorithms that we designed and developed throughout the research of this thesis.

The first proposed algorithm, the GCF, is greedy and prioritizes sending tasks to servers close to the client and with the lowest processing queue times. According to Section 6.3.2, the GCF reduced the workload execution time by up to 66.5 % compared to the fully local execution and up to 46.0 % compared to literature algorithms.

The other two algorithms use two more contextual parameters than the GCF: CPU capacity and known routes of vehicles. Also, they use different strategies. For example, the second algorithm, GTT, sequentially analyzes each task and assigns them to the best available server, local or remote. The update of the best server is done after each new task assignment. It is based mainly on three criteria: distance to the client and CPU capacity and availability. The GTT reduced the workload execution time by up to 73.9 % compared to fully local execution and up to 56.5 % compared to literature algorithms.

The third and last algorithm designed and developed in this research, BCV, is an intelligent algorithm based on the ABC metaheuristic. It scours the search space to find solutions with the best fitness scores. The BCV was the algorithm that presented the best performance in reducing execution time, being the best in 48 of 60 cases. It managed to reduce execution time by up to 75.6 % compared to fully local execution and up to 57.9 % compared to literature algorithms.

RQ3: How to avoid offloading failures and, ultimately, recover from them?

We propose that it be through the joint action of the framework and the proposed algorithms. In this case, any of the three proposed algorithms make task assignments with well-calculated risks of failure. These calculations are part of the restrictions described in the formulation of the problem and involve several contextual parameters, mainly energy and connectivity constraints. The first ensures that devices have enough energy to participate in computation offloading processes. In connectivity, client vehicles do mobility prediction calculations. After, they analyze whether they can send tasks, wait for their remote processing, and receive the results without losing connections with servers.

The experiments presented in Section 6.3.1 show that the GCF was able to offload up to 89.4 % of tasks successfully and only have 1.1 % failures. The GCF offloaded more tasks than other literature algorithms in 39 of 60 cases and have fewer failures in 38 of 60 cases. This performance was even better with the GTT and BCV. These two algorithms use special contextual information from known routes of vehicles that makes the mobility prediction calculation even

more accurate. Thus, the GTT successfully offloaded up to 90.8 % of the tasks and had only 0.1 % failures, outperforming the other algorithms in 56 of 60 cases. The BCV offloaded up to 91.4 % of tasks successfully and reached 0.0 % failures, outperforming the other algorithms in 55 of 60 cases.

Although the proposed algorithms have minimal failure rates, it is also necessary to have a failure recovery mechanism as a last resort. In this case, we designed and developed this mechanism in the proposed framework. Section 5.1 shows that the framework stores a backup of offloaded tasks and monitors for possible failures. When a failure is detected, the framework sends the copy of the lost task to local execution, preventing the application from being harmed.

7.2 General Discussion

This work achieved its main objective by presenting solutions that minimize the execution time of vehicular applications reliably through computation offloading in VEC systems. To do this, we followed each step listed in the specific objectives shown in Chapter 1. For example, we designed and implemented a VEC system with WAVE and 5G technologies simulating a real vehicular environment with vehicles, base stations, and edge servers. We designed and implemented a context-oriented framework to support and manage all stages of computation offloading processes, including a failure recovery mechanism. We also enabled the simulation of fully local executions of a vehicular application with different workloads.

Next, we investigated several state-of-the-art works in the area. We found that current solutions do not realize their full potential in terms of technology. Most works use only one communication technology, WAVE or cellular. When some work uses more than one technology, the authors consider only their sequential use. Also, most works use only one type of server, vehicle or edge server. Thus, there is underutilization of available computational resources and wasted opportunities for performance gains in computation offloading processes. In addition, such state-of-the-art works in the area do not use important contextual information or failure recovery mechanisms that would help make computation offloading more reliable. Finally, the algorithmic strategies used in these works have some disadvantages presented by other literature works and were not submitted to different vehicular environments for validation.

Unlike these works, our thesis explored all the available technological potential. For this, we used WAVE and 5G technologies simultaneously when sending or receiving tasks. We also used all available computing resources, both in VCs and Edge. In addition, we utilized a

failure recovery mechanism and additional contextual information to make offloading computation more efficient and reliable. Among this information, we highlight known routes of vehicles, which made mobility predictions more accurate and was used for the first time in computation offloading, to the best of our knowledge. In addition, we used different algorithmic strategies, including one based on the ABC metaheuristic.

Then, we evaluated and compared the proposed solutions and some literature solutions in different vehicular scenarios. We could see that our proposed solutions had the best performance. Thus, we presented three ways to solve the problem addressed in this thesis by minimizing the execution time of vehicular applications reliably in VEC systems. In particular, our best algorithm, BCV, showed up to a 75.6 % of reduction in execution time and 0.0 % failures.

These results are significant because they show that the computation offloading technique, when well managed, can reliably improve the performance of vehicular applications through other cooperating devices in the vehicular environment. This performance is even more essential for some applications that have critical latency requirements. A slight delay in the execution of computation tasks can compromise the usefulness of these applications and their data and even people's lives, as is the case with some applications of autonomous vehicles. Such vehicles and their complex applications are increasingly close to becoming part of our daily lives and are hungry for computational resources. Therefore, we believe that the work developed in this thesis can offer a promising alternative to enable the execution of these applications.

7.3 Future Work

The computation offloading in VEC systems has several aspects that can be researched and improved. Therefore, we have listed below some research directions that can extend and improve our work.

- Computation offloading through new technologies. New communication technologies have emerged and presented new opportunities, such as higher data rates, but they also present new challenges. In this sense, research can be done to adapt computation offloading processes to these new technologies.
- Multi-objective optimization. In our work, we consider only a single-objective problem to minimize the execution time of vehicular applications. However, the problem can be extended to, in addition to minimizing the execution time, also considering the minimization

of energy consumption and financial costs (including cellular charges).

- Inclusion of traditional cloud servers. We have not considered these servers in our work due to critical latency constraints. However, they can be considered in future work to handle more complex and non-delay-sensitive applications.
- Real-time adaptation of parameters. Some parameters can impact computation offloading processes and the performance of vehicular applications. We can mention parameters as transmission power, data rate, number of hops, channels and frequency bands, and antenna configuration.
- Improvement of vehicular mobility prediction. This prediction influences the reliability of computation offloading processes and involves complex calculations, including probabilities of direction changes. Bad predictions can result in failures, and good predictions help improve the performance of offloading processes.
- Adaptation of algorithms concerning 5G signal blocking. The 5G wireless signal is very susceptible to propagation blocks due to obstacles. Thus, we must consider mechanisms to deal with this problem.
- New experiments with new setups and scenarios. For example, can be used: scenarios with other radio propagation models, without full coverage of base stations and mmWave on highways, that not all vehicles have the necessary technologies or application codes, and even some real experiments. In addition, hybrid (electric and combustion) vehicles, buildings, and multiple clients can integrate the scenarios.
- Different system models and applications. Such models should consider real bandwidth (which varies), other radio propagation models and antennas, and server evaluations based on signal-to-noise ratio and reception power. Moreover, different vehicular applications have different requirements for latency, processing, and task priority levels.
- Privacy, security, and incentive issues. Risks such as exposing vehicular data to untrusted servers or tasks containing viruses in the offloading process must be addressed. Furthermore, motivating network nodes to share their computing resources is another challenge that needs further study.
- Experimental evaluation issues. New graphics can be considered, such as the number of failures by type (e.g., 5G, WAVE, and connectivity), energy consumption, transmission and execution delay, distance, link lifetime (real and estimated), and packet delivery ratio. Additionally, other solution strategies and benchmarks can be used for comparison.

BIBLIOGRAPHY

ABDELHAMID, S.; BENKOCZI, R.; HASSANEIN, H. S. Vehicular clouds: ubiquitous computing on wheels. In: **Emergent Computation**. [*S. l.*]: Springer, 2017. p. 435–452.

AHMED, B.; MALIK, A. W.; HAFEEZ, T.; AHMED, N. Services and simulation frameworks for vehicular cloud computing: A contemporary survey. **EURASIP Journal on Wireless Communications and Networking**, Springer, v. 2019, n. 1, p. 4, 2019.

AKHTAR, N.; ERGEN, S. C.; OZKASAP, O. Vehicle mobility and communication channel models for realistic and efficient highway vanet simulation. **IEEE Transactions on Vehicular Technology**, IEEE, v. 64, n. 1, p. 248–262, 2014.

AL-SULTAN, S.; AL-DOORI, M. M.; AL-BAYATTI, A. H.; ZEDAN, H. A comprehensive survey on vehicular ad hoc network. **Journal of network and computer applications**, Elsevier, v. 37, p. 380–392, 2014.

ALVES, R.; CAMPBELL, I.; COUTO, R.; CAMPISTA, M.; MORAES, I.; RUBINSTEIN, M.; COSTA, L.; DUARTE, O.; ABDALLA, M. Redes veiculares: Princípios, aplicações e desafios. **Minicursos do Simpósio Brasileiro de Redes de Computadores, SBRC**, p. 1–56, 2009.

ANSARI, K. Joint use of dsrc and c-v2x for v2x communications in the 5.9 ghz its band. **IET Intelligent Transport Systems**, Wiley Online Library, v. 15, n. 2, p. 213–224, 2021.

ARENA, F.; PAU, G.; SEVERINO, A. A review on ieee 802.11 p for intelligent transportation systems. **Journal of Sensor and Actuator Networks**, Multidisciplinary Digital Publishing Institute, v. 9, n. 2, p. 22, 2020.

ARS TECHNICA. **FCC takes spectrum from auto industry in plan to "supersize" Wi-Fi**. 2020. Last accessed March 2021. Available at: https://arstechnica.com/tech-policy/2020/11/fcc-adds-45mhz-to-wi-fi-promising-supersize-networks-on-5ghz-band/.

ASLAM, B.; ZOU, C. C. Optimal roadside units placement along highways. In: **2011 IEEE Consumer Communications and Networking Conference (CCNC)**. [*S. l.*]: IEEE, 2011. p. 814–815.

AUSTIN, P. C.; HUX, J. E. A brief note on overlapping confidence intervals. **Journal of vascular surgery**, Elsevier, v. 36, n. 1, p. 194–195, 2002.

AUTOMOTIVE ASSOCIATION 5GAA. **A Visionary Roadmap for Advanced Driving Use Cases Connectivity Technologies and Radio Spectrum Needs**. 2020. Last accessed April 2021. Available at: https://5gaa.org/wp-content/uploads/2020/09/A-Visionary-Roadmap-for-Advanced-Driving-Use-Cases-Connectivity-Technologies-and-Radio-Spectrum-Needs.pdf.

BARB, G.; OTESTEANU, M. 4g/5g: A comparative study and overview on what to expect from 5g. In: IEEE. **2020 43rd International Conference on Telecommunications and Signal Processing (TSP)**. [*S. l.*], 2020. p. 37–40.

BARH, D. **Artificial Intelligence in Precision Health: From Concept to Applications**. London: Academic Press, 2020.

BATTERY UNIVERSITY GROUP. **BU-804: How to Prolong Lead-acid Batteries**. 2019. Last accessed February 2021. Available at: https://batteryuniversity.com/learn/article/how_to_restore_and_prolong_lead_acid_batteries.

BATTERY UNIVERSITY GROUP. **BU-1003: Electric Vehicle (EV)**. 2020. Last accessed February 2021. Available at: https://batteryuniversity.com/learn/article/electric_vehicle_ev.

BELIA, S.; FIDLER, F.; WILLIAMS, J.; CUMMING, G. Researchers misunderstand confidence intervals and standard error bars. **Psychological methods**, American Psychological Association, v. 10, n. 4, p. 389, 2005.

BLOCHO, M. Heuristics, metaheuristics, and hyperheuristics for rich vehicle routing problems. In: **Smart Delivery Systems**. [*S. l.*]: Elsevier, 2020. p. 101–156.

BLOOMBERG LAW. **INSIGHT: Elaine Chao Needs to Better Prioritize DOT's Spectrum Fights**. 2019. Last accessed March 2021. Available at: https://news.bloomberglaw.com/tech-and-telecom-law/insight-elaine-chao-needs-to-better-prioritize-dots-spectrum-fights.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM computing surveys (CSUR)**, Acm New York, NY, USA, v. 35, n. 3, p. 268–308, 2003.

BOUKERCHE, A.; ROBSON, E. Vehicular cloud computing: Architectures, applications, and mobility. **Computer networks**, Elsevier, v. 135, p. 171–189, 2018.

BOUKERCHE, A.; SOTO, V. An efficient mobility-oriented retrieval protocol for computation offloading in vehicular edge multi-access network. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 21, n. 6, p. 2675–2688, 2020.

BOUKERCHE, A.; SOTORO, V. Computation offloading and retrieval for vehicular edge computing: Algorithms, model and classification. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 4, p. 1–35, 2020.

CHANDRAMOULI, D.; LIEBHART, R.; PIRSKANEN, J. **5G for the Connected World**. Hoboken: John Wiley & Sons, 2019.

CHARITOS, M.; KALIVAS, G. Mimo hetnet ieee 802.11p-lte deployment in a vehicular urban environment. **Vehicular Communications**, Elsevier, v. 9, p. 222–232, 2017.

CHEN, C.; CHEN, L.; LIU, L.; HE, S.; YUAN, X.; LAN, D.; CHEN, Z. Delay-optimized v2v-based computation offloading in urban vehicular edge computing and networks. **IEEE Access**, IEEE, v. 8, p. 18863–18873, 2020.

CHEN, M.; HAO, Y.; QIU, M.; SONG, J.; WU, D.; HUMAR, I. Mobility-aware caching and computation offloading in 5g ultra-dense cellular networks. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 16, n. 7, p. 974, 2016.

CHEN, T.; XIAO, R. Enhancing artificial bee colony algorithm with self-adaptive searching strategy and artificial immune network operators for global optimization. **The Scientific World Journal**, Hindawi, v. 2014, p. 1–12, 2014.

CUI, T.; HU, Y.; SHEN, B.; CHEN, Q. Task offloading based on lyapunov optimization for mec-assisted vehicular platooning networks. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 19, n. 22, p. 4974, 2019.

DREYER, N.; MOLLER, A.; MIR, Z. H.; FILALI, F.; KURNER, T. A data traffic steering algorithm for ieee 802.11 p/lte hybrid vehicular networks. In: IEEE. **2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)**. [*S. l.*], 2016. p. 1–6.

DU, J.; YU, F. R.; CHU, X.; FENG, J.; LU, G. Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 2, p. 1079–1092, 2018.

DUAN, X.; WANG, X.; LIU, Y.; ZHENG, K. Sdn enabled dual cluster head selection and adaptive clustering in 5g-vanet. In: IEEE. **2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)**. [*S. l.*], 2016. p. 1–5.

EICHLER, S. Performance evaluation of the ieee 802.11 p wave communication standard. In: IEEE. **2007 IEEE 66th Vehicular Technology Conference**. [*S. l.*], 2007. p. 2199–2203.

EUROPEAN COMMISSION. **Harmonisation of the 5.9 GHz spectrum band for real-time information exchange will improve road and urban rail transport safety**. 2020. Last accessed March 2021. Available at: https://ec.europa.eu/digital-single-market/en/news/harmonisation-59-ghz-spectrum-band-real-time-information-exchange-will-improve-road-and-urban.

EZUGWU, A. E.; ADELEKE, O. J.; AKINYELU, A. A.; VIRIRI, S. A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems. **Neural Computing and Applications**, Springer, v. 32, n. 10, p. 6207–6251, 2020.

FCC. **FCC Modernizes 5.9 GHz Band to Improve Wi-Fi and Automotive Safety**. 2020. Last accessed March 2021. Available at: https://docs.fcc.gov/public/attachments/DOC-368228A1.pdf.

FENG, J.; LIU, Z.; WU, C.; JI, Y. Ave: Autonomous vehicular edge computing framework with aco-based scheduling. **IEEE Transactions on Vehicular Technology**, IEEE, v. 66, n. 12, p. 10660–10675, 2017.

FENG, J.; LIU, Z.; WU, C.; JI, Y. Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. **IEEE Vehicular Technology Magazine**, IEEE, v. 14, n. 1, p. 28–36, 2018.

FIERCEWIRELESS. **FCC's O'Rielly: 5.9 GHz band is 'a mess'**. 2018. Last accessed March 2021. Available at: https://www.fiercewireless.com/wireless/fcc-s-o-rielly-5-9-ghz-band-a-mess.

FIERCEWIRELESS. **FCC moves to authorize C-V2X in 5.9 GHz band**. 2020. Last accessed March 2021. Available at: https://www.fiercewireless.com/regulatory/fcc-moves-to-authorize-c-v2x-5-9-ghz-band.

GAO, K. Z.; SUGANTHAN, P. N.; PAN, Q. K.; TASGETIREN, M. F.; SADOLLAH, A. Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion. **Knowledge-based systems**, Elsevier, v. 109, p. 1–16, 2016.

GIORDANI, M.; ZANELLA, A.; HIGUCHI, T.; ALTINTAS, O.; ZORZI, M. Performance study of lte and mmwave in vehicle-to-network communications. In: IEEE. **2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)**. [*S. l.*], 2018. p. 1–7.

GROUPE RENAULT. **How to prepare your electric car to be parked for an extended period of time**. 2020. Last accessed February 2021. Available at: https://easyelectriclife.groupe.renault.com/en/day-to-day/charging/how-to-prepare-your-electric-car-to-be-parked-for-an-extended-period-of-time/.

GU, B.; ZHOU, Z. Task offloading in vehicular mobile edge computing: A matching-theoretic framework. **IEEE Vehicular Technology Magazine**, IEEE, v. 14, n. 3, p. 100–106, 2019.

GU, X.; ZHANG, G. Energy-efficient computation offloading for vehicular edge computing networks. **Computer Communications**, Elsevier, v. 166, p. 244–253, 2021.

GUAN, S. **Efficient and Proactive Offloading Techniques for Sustainable and Mobility-aware Resource Management in Heterogeneous Mobile Cloud Environments**. Thesis (PhD) – Université d'Ottawa/University of Ottawa, 2020.

GUO, H.; ZHANG, J.; LIU, J. Fiwi-enhanced vehicular edge computing networks: Collaborative task offloading. **IEEE Vehicular Technology Magazine**, IEEE, v. 14, n. 1, p. 45–53, 2018.

HÄRRI, J.; BONNET, C.; FILALI, F. Kinetic mobility management applied to vehicular ad hoc network protocols. **Computer Communications**, Elsevier, v. 31, n. 12, p. 2907–2924, 2008.

HARTENSTEIN, H.; LABERTEAUX, K. A tutorial survey on vehicular ad hoc networks. **Communications Magazine, IEEE**, IEEE, v. 46, n. 6, p. 164–171, 2008.

HASSIJA, V.; SAXENA, V.; CHAMOLA, V. A mobile data offloading framework based on a combination of blockchain and virtual voting. **Software: Practice and Experience**, Wiley Online Library, p. 1–18, 2020.

HE, Z.-a.; MA, C.; WANG, X.; LI, L.; WANG, Y.; ZHAO, Y.; GUO, H. A modified artificial bee colony algorithm based on search space division and disruptive selection strategy. **Mathematical problems in engineering**, Hindawi, v. 2014, p. 1–14, 2014.

HEISE MEDIEN. **Autovernetzung: FCC stoppt WLANp und schiebt C-V2X an**. 2020. Last accessed March 2021. Available at: https://www.heise.de/news/Autovernetzung-FCC-stoppt-WLANp-und-schiebt-C-V2X-an-4976068.html.

HOLDING, J. **What happens when an electric car runs out?** 2020. Last accessed February 2021. Available at: https://www.drivingelectric.com/your-questions-answered/320/what-happens-when-electric-car-runs-out.

HONG, K.; XING, D.; RAI, V.; KENNEY, J. Characterization of dsrc performance as a function of transmit power. In: **Proceedings of the sixth ACM international workshop on VehiculAr InterNETworking**. [*S. l.*: *s. n.*], 2009. p. 63–68.

HOU, X.; LI, Y.; CHEN, M.; WU, D.; JIN, D.; CHEN, S. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. **IEEE Transactions on Vehicular Technology**, IEEE, v. 65, n. 6, p. 3860–3873, 2016.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, p. eprint, 2017.

HUANG, X.; XU, K.; LAI, C.; CHEN, Q.; ZHANG, J. Energy-efficient offloading decision-making for mobile edge computing in vehicular networks. **EURASIP Journal on Wireless Communications and Networking**, Springer, v. 2020, n. 1, p. 35, 2020.

IEEE. Ieee family of standards for wireless access in vehicular environments. In: **IEEE 1609 series**. New York: IEEE, 2011.

IHS MARKIT. **These OEMs are launching 5G-enabled cars years before the tech goes mainstream**. 2020. Last accessed April 2021. Available at: https://ihsmarkit.com/research-analysis/these-oems-are-launching-5genabled-cars-years-before-the-tech-.html.

INTEL. **Intel® Celeron® Processor 1.30 GHz, 256K Cache, 100 MHz FSB**. 2021. Last accessed February 2021. Available at: https://ark.intel.com/content/www/us/en/ark/products/27169/intel-celeron-processor-1-30-ghz-256k-cache-100-mhz-fsb.html.

INTEL. **Intel® Xeon® Processor 2.60 GHz, 512K Cache, 400 MHz FSB**. 2021. Last accessed February 2021. Available at: https://ark.intel.com/content/www/us/en/ark/products/27271/intel-xeon-processor-2-60-ghz-512k-cache-400-mhz-fsb.html.

INTERNATIONAL RAILWAY JOURNAL. **Europe boosts spectrum allocation for Intelligent Transport Systems**. 2020. Last accessed March 2021. Available at: https://www.railjournal.com/telecoms/europe-boosts-spectrum-allocation-for-intelligent-transport-systems/.

JIANG, D.; DELGROSSI, L. Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments. In: IEEE. **Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE**. [*S. l.*], 2008. p. 2036–2040.

JONSSON, P.; DAVIS, S.; LINDER, P.; GOMROKI, A.; ZAIDI, A. *et al.* Ericsson mobility report. **Ericsson Mobility Report**, p. 1–36, November 2020.

KANG, J.; YU, R.; HUANG, X.; WU, M.; MAHARJAN, S.; XIE, S.; ZHANG, Y. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 3, p. 4660–4670, 2018.

KARABOGA, D. An idea based on honey bee swarm for numerical optimization. **Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department**, Kayseri, p. 1–10, 2005.

KARABOGA, D.; AKAY, B. A comparative study of artificial bee colony algorithm. **Applied mathematics and computation**, Elsevier, v. 214, n. 1, p. 108–132, 2009.

KHOSRAVANIAN, R.; MANSOURI, V.; WOOD, D. A.; ALIPOUR, M. R. A comparative study of several metaheuristic algorithms for optimizing complex 3-d well-path designs. **Journal of Petroleum Exploration and Production Technology**, Springer, v. 8, n. 4, p. 1487–1503, 2018.

KRAJZEWICZ, D. Traffic simulation with sumo – simulation of urban mobility. In: **Fundamentals of traffic simulation**. New York: Springer, 2010. p. 269–293.

KRATSIOS, M. **Emerging Technologies and their Expected Impact on Non-Federal Spectrum Demand**. 2019. Last accessed April 2021. Available at: https://trumpwhitehouse.archives.gov/wp-content/uploads/2019/05/Emerging-Technologies-and-Impact-on-Non-Federal-Spectrum-Demand-Report-May-2019.pdf.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

KUNCHE, P.; REDDY, K. **Metaheuristic Applications to Speech Enhancement**. Cham: Springer, 2016.

LAUKKONEN, J. **Car Batteries Are Made to Die**. 2019. Last accessed February 2021. Available at: https://www.lifewire.com/car-batteries-are-made-to-die-534765.

LI, B.; PEI, Y.; WU, H.; LIU, Z.; LIU, H. Computation offloading management for vehicular ad hoc cloud. In: SPRINGER. **International Conference on Algorithms and Architectures for Parallel Processing**. [*S. l.*], 2014. p. 728–739.

LI, C.; ZHANG, Y.; LUAN, T. H.; FU, Y. Building transmission backbone for highway vehicular networks: Framework and analysis. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 9, p. 8709–8722, 2018.

LI, H.; LI, X.; WANG, W. Joint optimization of computation cost and delay for task offloading in vehicular fog networks. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 31, n. 2, p. e3818, 2020.

LI, X.; DANG, Y.; AAZAM, M.; PENG, X.; CHEN, T.; CHEN, C. Energy-efficient computation offloading in vehicular edge cloud computing. **IEEE Access**, IEEE, v. 8, p. 37632–37644, 2020.

LIN, C.-C.; DENG, D.-J.; YAO, C.-C. Resource allocation in vehicular cloud computing systems with heterogeneous vehicles and roadside units. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 5, p. 3692–3700, 2017.

LIU, Y.; WANG, S.; ZHAO, Q.; DU, S.; ZHOU, A.; MA, X.; YANG, F. Dependency-aware task scheduling in vehicular edge computing. **IEEE Internet of Things Journal**, IEEE, v. 7, n. 6, p. 4961–4971, 2020.

L'HEUREUX, A.; GROLINGER, K.; ELYAMANY, H. F.; CAPRETZ, M. A. Machine learning with big data: Challenges and approaches. **Ieee Access**, IEEE, v. 5, p. 7776–7797, 2017.

MALANDRINO, F.; CASETTI, C.; CHIASSERINI, C.-F.; SOMMER, C.; DRESSLER, F. The role of parked cars in content downloading for vehicular networks. **IEEE Transactions on Vehicular Technology**, IEEE, v. 63, n. 9, p. 4606–4617, 2014.

MARSCH, P.; BULAKCI, Ö.; QUESETH, O.; BOLDI, M. **5G system design: architectural and functional considerations and long term research**. Hoboken: John Wiley & Sons, 2018.

MENOUAR, H.; LENARDI, M.; FILALI, F. Movement prediction-based routing (mopr) concept for position-based routing in vehicular networks. In: IEEE. **Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th**. [*S. l.*], 2007. p. 2101–2105.

MEZZAVILLA, M.; ZHANG, M.; POLESE, M.; FORD, R.; DUTTA, S.; RANGAN, S.; ZORZI, M. End-to-end simulation of 5g mmwave networks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 20, n. 3, p. 2237–2263, 2018.

MIDYA, S.; ROY, A.; MAJUMDER, K.; PHADIKAR, S. Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach. **Journal of Network and Computer Applications**, Elsevier, v. 103, p. 58–84, 2018.

MISRA, S.; BERA, S. Soft-van: Mobility-aware task offloading in software-defined vehicular network. **IEEE Transactions on Vehicular Technology**, IEEE, v. 69, n. 2, p. 2071–2078, 2019.

NABI, M.; BENKOCZI, R.; ABDELHAMID, S.; HASSANEIN, H. S. Resource assignment in vehicular clouds. In: IEEE. **2017 IEEE International Conference on Communications (ICC)**. [*S. l.*], 2017. p. 1–6.

NAIK, G.; CHOUDHURY, B.; PARK, J.-M. Ieee 802.11bd & 5g nr v2x: Evolution of radio access technologies for v2x communications. **IEEE Access**, IEEE, v. 7, p. 70169–70184, 2019.

NAMBOODIRI, V.; GAO, L. Prediction-based routing for vehicular ad hoc networks. **IEEE Transactions on Vehicular Technology**, IEEE, v. 56, n. 4, p. 2332–2345, 2007.

NGO, D. T.; NGUYEN, D. H. N.; LE-NGOC, T. Intercell interference coordination: Towards a greener cellular network. **Handbook of Green Information and Communication Systems**, p. 147–182, 2013.

NING, Z.; WANG, X.; HUANG, J. Mobile edge computing-enabled 5g vehicular networks: Toward the integration of communication and computing. **IEEE Vehicular Technology Magazine**, IEEE, v. 14, n. 1, p. 54–61, 2018.

NS-3 TEAM. **YansWifiPhy Class Reference**. 2021. Last accessed February 2021. Available at: https://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_phy.html.

NUTS AND VOLTS MAGAZINE. **Connected cars are coming**. 2018. Last accessed March 2021. Available at: https://www.nutsvolts.com/magazine/article/connected-cars-are-coming.

ORGANIZATION 5G AMERICAS. **Understanding mmWave Spectrum for 5G Networks**. 2020. Last accessed April 2021. Available at: https://www.5gamericas.org/wp-content/uploads/2020/12/InDesign-Understanding-mmWave-for-5G-Networks.pdf.

PANICHPAPIBOON, S.; PATTARA-ATIKOM, W. Connectivity Requirements for Self-Organizing Traffic Information Systems. **IEEE Transactions on Vehicular Technology**, v. 57, n. 6, p. 3333–3340, 2008.

PENTTINEN, J. **5G Explained - Security and Deployment of Advanced Mobile Communications**. Hoboken: Wiley Online Library, 2019.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. **IEEE communications surveys & tutorials**, IEEE, v. 16, n. 1, p. 414–454, 2013.

PHAM, Q.-V.; FANG, F.; HA, V. N.; PIRAN, M. J.; LE, M.; LE, L. B.; HWANG, W.-J.; DING, Z. A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. **IEEE Access**, IEEE, v. 8, p. 116974–117017, 2020.

PHAM, X.-Q.; NGUYEN, T.-D.; NGUYEN, V.; HUH, E.-N. Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing. **Symmetry**, Multidisciplinary Digital Publishing Institute, v. 11, n. 1, p. 58, 2019.

POLESE, M. Performance comparison of dual connectivity and hard handover for lte-5g tight integration in mmwave cellular networks. **arXiv preprint arXiv:1607.04330**, 2016.

QIAO, G.; LENG, S.; ZHANG, K.; HE, Y. Collaborative task offloading in vehicular edge multi-access networks. **IEEE Communications Magazine**, IEEE, v. 56, n. 8, p. 48–54, 2018.

RAHMAN, A. U.; MALIK, A. W.; SATI, V.; CHOPRA, A.; RAVANA, S. D. Context-aware opportunistic computing in vehicle-to-vehicle networks. **Vehicular Communications**, Elsevier, v. 24, p. 100236, 2020.

RAPPAPORT, T. S.; XING, Y.; MACCARTNEY, G. R.; MOLISCH, A. F.; MELLIOS, E.; ZHANG, J. Overview of millimeter wave communications for fifth-generation (5g) wireless networks—with a focus on propagation models. **IEEE Transactions on Antennas and Propagation**, v. 65, n. 12, p. 6213–6230, 2017.

RAZA, S.; LIU, W.; AHMED, M.; ANWAR, M. R.; MIRZA, M. A.; SUN, Q.; WANG, S. An efficient task offloading scheme in vehicular edge computing. **Journal of Cloud Computing**, Springer, v. 9, p. 1–14, 2020.

RAZA, S.; WANG, S.; AHMED, M.; ANWAR, M. R. A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions. **Wireless Communications and Mobile Computing**, Hindawi, v. 2019, p. 1–19, 2019.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **arXiv preprint arXiv:1804.02767**, p. eprint, 2018.

REGO, P. **Applying Smart Decisions, Adaptive Monitoring and Mobility Support for Enhancing Offloading Systems**. Thesis (PhD) – Universidade Federal do Ceará, 2016.

REGO, P. A.; COSTA, P. B.; COUTINHO, E. F.; ROCHA, L. S.; TRINTA, F. A.; SOUZA, J. N. de. Performing computation offloading on multiple platforms. **Computer Communications**, Elsevier, v. 105, p. 1–13, 2017.

REIS, A. B.; SARGENTO, S.; TONGUZ, O. K. Parked cars are excellent roadside units. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 18, n. 9, p. 2490–2502, 2017.

REIS, A. B.; SARGENTO, S.; TONGUZ, O. K. Smarter cities with parked cars as roadside units. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 19, n. 7, p. 2338–2352, 2018.

RILEY, G. F.; HENDERSON, T. R. The ns-3 network simulator. In: **Modeling and tools for network simulation**. [*S. l.*]: Springer, 2010. p. 15–34.

SANGUESA, J. A.; NARANJO, F.; TORRES-SANZ, V.; FOGUE, M.; GARRIDO, P.; MARTINEZ, F. J. On the study of vehicle density in intelligent transportation systems. **Mobile Information Systems**, Hindawi, v. 2016, p. 1–13, 2016.

SHAFI, M.; MOLISCH, A. F.; SMITH, P. J.; HAUSTEIN, T.; ZHU, P.; SILVA, P. D.; TUFVESSON, F.; BENJEBBOUR, A.; WUNDER, G. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. **IEEE journal on selected areas in communications**, IEEE, v. 35, n. 6, p. 1201–1221, 2017.

SHAHAM, S.; DING, M.; KOKSHOORN, M.; LIN, Z.; DANG, S.; ABBAS, R. Fast channel estimation and beam tracking for millimeter wave vehicular communications. **IEEE Access**, IEEE, v. 7, p. 141104–141118, 2019.

SHARIFI, M.; KAFAIE, S.; KASHEFI, O. A Survey and Taxonomy of Cyber Foraging of Mobile Devices. **IEEE Communications Surveys & Tutorials**, v. 14, n. 4, p. 1232–1243, 2012.

SHESKIN, D. J. **Handbook of parametric and nonparametric statistical procedures**. Boca Raton: Chapman and Hall/CRC, 2000.

SHIN, Y.; CHOI, H.; NAM, Y.; LEE, E. Data delivery protocol using the trajectory information on a road map in vanets. **Ad Hoc Networks**, Elsevier, v. 107, p. 102260, 2020.

SOMMER, C.; DRESSLER, F. **Vehicular networking**. Cambridge: Cambridge University Press, 2014.

SONG, J.; WU, Y.; XU, Z.; LIN, X. Research on car-following model based on sumo. In: IEEE. **The 7th IEEE/International Conference on Advanced Infocomm Technology**. [*S. l.*], 2014. p. 47–55.

SOUZA, A. B.; CELESTINO, J.; XAVIER, F. A.; OLIVEIRA, F. D.; PATEL, A.; LATIFI, M. Stable multicast trees based on ant colony optimization for vehicular ad hoc networks. In: IEEE. **The International Conference on Information Networking 2013 (ICOIN)**. [*S. l.*], 2013. p. 101–106.

SOUZA, A. B. D.; REGO, P. A.; CARNEIRO, T.; RODRIGUES, J. D. C.; FILHO, P. P. R.; SOUZA, J. N. D.; CHAMOLA, V.; ALBUQUERQUE, V. H. C. D.; SIKDAR, B. Computation offloading for vehicular environments: A survey. **IEEE Access**, IEEE, v. 8, p. 198214–198243, 2020.

STORCK, C. R.; DUARTE-FIGUEIREDO, F. A 5g v2x ecosystem providing internet of vehicles. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 19, n. 3, p. 550, 2019.

SUN, F.; HOU, F.; CHENG, N.; WANG, M.; ZHOU, H.; GUI, L.; SHEN, X. Cooperative task scheduling for computation offloading in vehicular cloud. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 11, p. 11049–11061, 2018.

SUN, J.; GU, Q.; ZHENG, T.; DONG, P.; QIN, Y. Joint communication and computing resource allocation in vehicular edge computing. **International Journal of Distributed Sensor Networks**, SAGE Publications Sage UK: London, England, v. 15, n. 3, p. 1550147719837859, 2019.

SUN, Y.; GUO, X.; SONG, J.; ZHOU, S.; JIANG, Z.; LIU, X.; NIU, Z. Adaptive learning-based task offloading for vehicular edge computing systems. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 4, p. 3061–3074, 2019.

SUN, Y.; SONG, J.; ZHOU, S.; GUO, X.; NIU, Z. Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach. In: IEEE. **2018 IEEE Global Communications Conference (GLOBECOM)**. [*S. l.*], 2018. p. 1–7.

TALBI, E.-G. **Metaheuristics: from design to implementation**. Hoboken: John Wiley & Sons, 2009. v. 74.

TAN, L. T.; HU, R. Q.; HANZO, L. Twin-timescale artificial intelligence aided mobility-aware edge caching and computing in vehicular networks. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 4, p. 3086–3099, 2019.

TEHRANI-MOAYYED, M.; RESTUCCIA, F.; BASAGNI, S. Comparative performance evaluation of mmwave 5g nr and lte in a campus scenario. **Proceedings of IEEE VTC 2020 Fall**, 2020.

TONGUZ, O.; WISITPONGPHAN, N.; BAI, F.; MUDALIGE, P.; SADEKAR, V. Broadcasting in VANET. **2007 Mobile Networking for Vehicular Environments**, p. 7–12, 2007.

UCAR, S.; ERGEN, S. C.; OZKASAP, O. Multihop-cluster-based ieee 802.11 p and lte hybrid architecture for vanet safety message dissemination. **IEEE Transactions on Vehicular Technology**, IEEE, v. 65, n. 4, p. 2621–2636, 2015.

UZCATEGUI, R.; ACOSTA-MARUM, G. Wave: a tutorial. **Communications Magazine, IEEE**, IEEE, v. 47, n. 5, p. 126–133, 2009.

VERMA, J. **Repeated measures design for empirical researchers**. Hoboken: John Wiley & Sons, 2015.

VIRIYASITAVAT, W.; TONGUZ, O. K.; BAI, F. Uv-cast: an urban vehicular broadcast protocol. **IEEE Communications Magazine**, IEEE, v. 49, n. 11, p. 116–124, 2011.

WANG, H.; LI, X.; JI, H.; ZHANG, H. Federated offloading scheme to minimize latency in mec-enabled vehicular networks. In: IEEE. **2018 IEEE Globecom Workshops (GC Wkshps)**. [*S. l.*], 2018. p. 1–6.

WANG, J.; FENG, D.; ZHANG, S.; TANG, J.; QUEK, T. Q. Computation offloading for mobile edge computing enabled vehicular networks. **IEEE Access**, IEEE, v. 7, p. 62624–62632, 2019.

WANG, Y.; LANG, P.; TIAN, D.; ZHOU, J.; DUAN, X.; CAO, Y.; ZHAO, D. A game-based computation offloading method in vehicular multi-access edge computing networks. **IEEE Internet of Things Journal**, IEEE, 2020.

WANG, Z.; ZHONG, Z.; ZHAO, D.; NI, M. Vehicle-based cloudlet relaying for mobile computation offloading. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 11, p. 11181–11191, 2018.

WEVERS, K.; LU, M. V2x communication for its-from ieee 802.11 p towards 5g. **IEEE 5G Tech Focus**, v. 1, n. 2, p. 5–10, 2017.

WHAIDUZZAMAN, M.; SOOKHAK, M.; GANI, A.; BUYYA, R. A survey on vehicular cloud computing. **Journal of Network and Computer Applications**, Elsevier, v. 40, p. 325–344, 2014.

WORLD HEALTH ORGANIZATION. **Number of registered vehicles**. 2020. Last accessed June 2021. Available at: https://www.who.int/data/gho/data/indicators/indicator-details/GHO/number-of-registered-vehicles.

WU, Q.; GE, H.; LIU, H.; FAN, Q.; LI, Z.; WANG, Z. A task offloading scheme in vehicular fog and cloud computing system. **IEEE Access**, IEEE, v. 8, p. 1173–1184, 2019.

XU, D.; LI, Y.; CHEN, X.; LI, J.; HUI, P.; CHEN, S.; CROWCROFT, J. A survey of opportunistic offloading. **IEEE Communications Surveys & Tutorials**, IEEE, v. 20, n. 3, p. 2198–2236, 2018.

YE, H.; LIANG, L.; LI, G. Y.; KIM, J.; LU, L.; WU, M. Machine learning for vehicular networks: Recent advances and application examples. **IEEE Vehicular Technology Magazine**, IEEE, v. 13, n. 2, p. 94–101, 2018.

YOUSAFZAI, A.; YAQOOB, I.; IMRAN, M.; GANI, A.; NOOR, R. M. Process migration-based computational offloading framework for iot-supported mobile edge/cloud computing. **IEEE Internet of Things Journal**, IEEE, v. 7, n. 5, p. 4171–4182, 2019.

YOUSEFI, S.; MOUSAVI, M.; FATHY, M. Vehicular ad hoc networks (VANETs): challenges and perspectives. In: **ITS Telecommunications Proceedings, 2006 6th International Conference on**. [*S. l.*: *s. n.*], 2006. p. 761–766.

YOUSEFPOUR, A.; FUNG, C.; NGUYEN, T.; KADIYALA, K.; JALALI, F.; NIAKANLAHIJI, A.; KONG, J.; JUE, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, Elsevier, v. 98, p. 289–330, 2019.

YÜRÜR, Ö.; LIU, C. H.; SHENG, Z.; LEUNG, V. C.; MORENO, W.; LEUNG, K. K. Context-awareness for mobile sensing: A survey and future directions. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 1, p. 68–93, 2014.

ZHANG, J.; GUO, H.; LIU, J.; ZHANG, Y. Task offloading in vehicular edge computing networks: A load-balancing solution. **IEEE Transactions on Vehicular Technology**, IEEE, v. 69, n. 2, p. 2092–2104, 2019.

ZHANG, J.; LETAIEF, K. B. Mobile edge intelligence and computing for the internet of vehicles. **Proceedings of the IEEE**, IEEE, v. 108, n. 2, p. 246–261, 2019.

ZHANG, M.-D.; ZHAN, Z.-H.; LI, J.-J.; ZHANG, J. Tournament selection based artificial bee colony algorithm with elitist strategy. In: SPRINGER. **International Conference on Technologies and Applications of Artificial Intelligence**. [*S. l.*], 2014. p. 387–396.

ZHANG, X.; ZHANG, J.; LIU, Z.; CUI, Q.; TAO, X.; WANG, S. Mdp-based task offloading for vehicular edge computing under certain and uncertain transition probabilities. **IEEE Transactions on Vehicular Technology**, IEEE, v. 69, n. 3, p. 3296–3309, 2020.

ZHAO, L.; QU, S.; YI, Y. A modified cluster-head selection algorithm in wireless sensor networks based on leach. **EURASIP Journal on Wireless Communications and Networking**, SpringerOpen, v. 2018, n. 1, p. 1–8, 2018.

ZHOU, S.; SUN, Y.; JIANG, Z.; NIU, Z. Exploiting moving intelligence: Delay-optimized computation offloading in vehicular fog networks. **IEEE Communications Magazine**, IEEE, v. 57, n. 5, p. 49–55, 2019.

ZHOU, Z.; LIU, P.; CHANG, Z.; XU, C.; ZHANG, Y. Energy-efficient workload offloading and power control in vehicular edge computing. In: IEEE. **2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)**. [*S. l.*], 2018. p. 191–196.

ZHU, C.; TAO, J.; PASTOR, G.; XIAO, Y.; JI, Y.; ZHOU, Q.; LI, Y.; YLÄ-JÄÄSKI, A. Folo: Latency and quality optimized task allocation in vehicular fog computing. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 3, p. 4150–4161, 2018.

ZHU, L.; LI, C.; LI, B.; WANG, X.; MAO, G. Geographic routing in multilevel scenarios of vehicular ad hoc networks. **IEEE Transactions on Vehicular Technology**, IEEE, v. 65, n. 9, p. 7740–7753, 2016.

ZOBOLAS, G.; TARANTILIS, C. D.; IOANNOU, G. Exact, heuristic and meta-heuristic algorithms for solving shop scheduling problems. In: **Metaheuristics for scheduling in industrial and manufacturing applications**. [*S. l.*]: Springer, 2008. p. 1–40.